Institut für parallele und verteilte Systeme

Abteilung für verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Studienarbeit Nr. 2439

Implementierung von Reglern und Steuerelementen für ein Android-basiertes E-Learning-Spiel für Smartphones

Michael Merg

Studiengang:	Dipl. Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Betreuer:	IST: Patrick Weber, Andrei Kramer
begonnen am:	01.09.2013
beendet am:	29.01.2013
CR-Klassifikation:	G.1.3, H.5.2, I.2.9, J.2

Inhaltsverzeichnis

1 Einleitung

Es gibt wohl keine angenehmere Art zu lernen als mit einem Spiel. Mit diesem Gedanke wurde am Institut für Systemtheorie und Regelungstechnik das Spiel SubmarineRT auf Basis von Adobe Flash entwickelt. In verschiedenen Lektionen soll Schritt für Schritt durch den Regelentwurf geleitet werden. Allerdings ist die Steuerung mit Maus und Tastatur weder sehr intuitiv noch kommt viel Spielspaß dabei auf.

Moderne Apps auf Smartphones und Tablets bieten in dieser Hinsicht ganz andere Möglichkeiten. Durch verschiedenste Sensoren und einem großen Touchscreen-Display ist die Bedienung vieler Apps sehr vielfältig und intuitiv.

Das Ziel dieser Arbeit ist die Vereinigung dieser neuen Möglichkeiten mit dem bisherigen Spielkonzept. Anstelle des U-Boots im SubmarineRT Spiel steht im Spiel SpaceController ein kleines Raumschiff im Mittelpunkt. Wie auch beim U-Boot Spiel muss das Raumschiff bestmöglichst einer vorgegeben Bahnkurve (Trajektorie) folgen. Neben der manuellen Steuerung mittels Lagesensor stehen drei weitere Spielmodi zur Verfügung. Im ersten Modus muss ein Zustandsregler entworfen werden. Im zweiten wird dieser noch durch einen Beobachter erweitert. Im letzten Modus verfolgt das Raumschiff per Loop-Shaping den Verlauf der Trajektorie. Durch die Kombination von der Steuerung mit dem Lagesensor und der Regelung ist noch mehr Spielspaß garantiert.

Thematisch ist die App in zwei Bereiche aufgeteilt wodurch zwei Studienarbeiten entstanden sind. In dieser Arbeit liegt der Fokus auf den regelungstechnischen Hintergründen des Spiels, der Steuerung über die von Android bereitgestellten Orientierungssensoren und der Menüführung des Spiels. Die Visualisierung der Spieloberfläche und deren effiziente Verwaltung ist Thema der Studienarbeit [Her13] von Dominik Herr. Des Weiteren wird in der Arbeit von Herr auf die Verwaltung des Highscore-Mechanismus des Spiels eingegangen. In beiden Arbeiten werden jeweils die Schnittstellen zwischen diesen einzelnen Komponenten beschrieben. Durch die Aufspaltung in diese einzelnen Module ist es sehr leicht das Spiel mit weitere Levels, neue Spielmodi und neuen Funktionen erweitert werden.

2 Grundlagen

Da das entwickelte Spiel die Konzepte der Regelungstechnik einfach und sehr anschaulich auf einem mobilen Endgerät zeigen soll, werden in diesem Kapitel zunächst die wesentlichen Unterschiede einer Steuerung und einer Regelung beschrieben. Anschließend wird der Aufbau und die Funktionsweise des Regelkreises erläutert.

Im zweiten Teil dieses Kapitels wird der Aufbau des Betriebssystems Android beschrieben und ein kurzer Überblick über die Entwicklung von mobilen Anwendungen (Apps) gegeben.

2.1 Regelungstechnik

Die Regelungstechnik befasst sich mit der Regelung bzw. der Beeinflussung des dynamischen Verhaltens eines Systems. Ein dynamisches System ist eine signalverarbeitende Funktionseinheit, wobei die Systemeingänge (Ursache) und die zeitliche Auswirkung in Relation zueinander gebracht werden [Unbo8].

Ein aus dem Alltag bekanntes Beispiel ist die Heizung eines Hauses. Bei der außentemperaturgesteuerten Heizung wird mittels eines Temperaturfühlers die Außentemperatur gemessen. Das Steuergerät der Heizung steuert anhand des gemessenen Wertes den Fluss des heißen Wassers im gesamten Haus. Ein gravierender Nachteil einer solchen Anlage ist, dass auf unerwartete Störungen wie etwa auf ein offenes Fenster nicht regiert werden kann. Grund dafür ist, dass der Heizung der Außentemperaturfühler als einzige Steuergröße zur Verfügung steht.

Anders ist dies bei einer Heizung die nicht gesteuert, sondern geregelt wird. Mit einem Raumtemperaturregler wird die gewünschte Temperatur (z.B. 22 °C) in einem Führungsraum eingestellt. Weicht die gewünschte Temperatur (Führungsgröße) von der tatsächlichen Temperatur in diesem Raum ab, d.h. die aktuelle Raumtemperatur ist z.B. 18 °C oder sie ist sogar höher als die gewünschte Temperatur, so wird mehr heißes Wasser in die Heizkörper geleitet bzw. der Fluss des Wassers verringert. Als Folge wird die Raumtemperatur so lange erhöht, bis sie dem gewünschten Wert, welcher vom Nutzer festgelegt wurde, entspricht. Die Differenz zwischen der Raumtemperatur und der eingestellten Soll-Temperatur regelt dabei den Fluss des heißen Wassers. Tritt jetzt wie bei der gesteuerten Heizung eine unerwartete Störung, etwa durch eine geöffnetes Fenster, auf, wird diese Änderung umgehend vom Raumtemperaturfühler erkannt und der Fluss des heißen Wassers so angepasst, dass die Raumtemperatur wieder der gewünschten Temperatur entspricht.

Im Gegensatz zu einer Steuerung wird bei der Regelung ständig die Ausgangsgröße zurückgeführt, um aus der Differenz von Ist- und Sollwert die Regelabweichung zu bestimmen. Durch die Rückführung der Ausgangsgröße entsteht ein geschlossener Kreis

der als *Regelkreis* bezeichnet wird. Die Funktionsweise des Regelkreises lässt sich einfach an einem Blockdiagramm, wie in Abbildung 2.1 dargestellt, veranschaulichen.

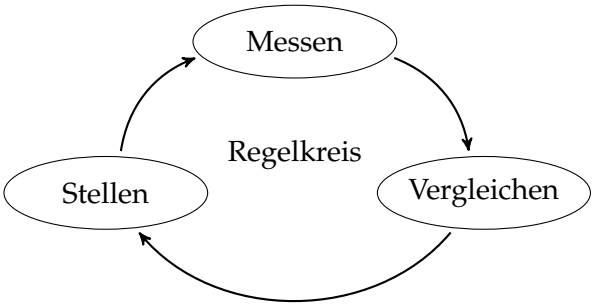
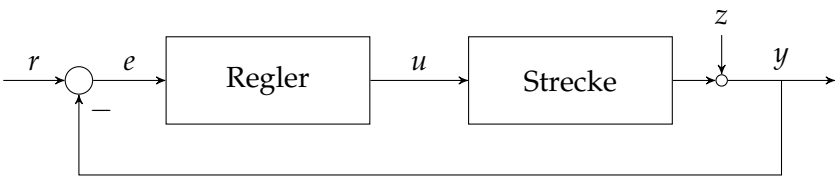


Abbildung 2.1: Schematische Darstellung eines Regelkreis

Wie im Eingangsbeispiel beschrieben wird die zu regelnde Größe (Regelgröße) gemessen und anschließend mit der Führungsgröße verglichen. Die darauf resultierende Differenz, die Regelabweichung, wird vom Regler verwendet, um die Stellgröße der Regelstrecke zu bestimmt. Die Stellgröße wirkt auf das System, wodurch sich die resultierende Regelgröße ändert. Die Regelgröße wird anschließen erneut gemessen und somit entsteht ein geschlossener Kreislauf.

2.1.1 Regelkreis

Der entstandene Regelkreis ist nicht nur auf unser Beispiel zutreffend, sondern jeder Regelkreis ist nach dem selben Schema aufgebaut. Dabei ist es egal ob es sich um ein mechanisches, hydraulisches oder elektrisches Übertragungssystem handelt. Auch in der Biologie werden biologische Übertragungssysteme mit Hilfe dieses Regelkreises analysiert. Dieser Regelkreis wird in der Systemtheorie als *Standard-Regelkreis* bezeichnet. Der Standard-Regelkreis wird oft mit einem Blockschaltbild (Abbildung 2.2) dargestellt.



r : Führungsgröße
e : Regelabweichung
u : Stellgröße
z : Störgröße
y : Regelgröße

Abbildung 2.2: Blockschaltbild eines Standardregelkreis

Die Regelstrecke, kurz Strecke, ist der wichtigste Teil des Regelkreises. Sie stellt das zu regelnde dynamische System dar, der ein gewisses Verhalten eingeprägt wird. Dabei soll die Ausgangsgröße y, auch Regelgröße genannt, trotz der Einwirkung einer Störgröße z, der Führungsgröße r bestmöglich folgen.

Um das zeitliche Verhalten einer Stecke erfassen zu können wird auf die Eingangsgröße r, auch Stellgröße genannt, einen sprunghafte Veränderung gegeben. Dabei wird die Ausgangsgröße y des Systems beobachtet. Die Ausgabe des Systems auf eine solche sprunghafte Änderung der Stellgröße wird Sprungantwort genannt. Durch die Auswertung der Sprungantwort können bestimmte, für die Reglerdimensionierung benötigte, Kenngrößen abgelesen werden.

Der Regler hat die Aufgabe die Regelgröße y zu messen, sie mit der Führungsgröße r zu vergleichen und bei Differenz die Stellgröße u so zu verändern, dass Soll- und Istwert möglichst genau übereinstimmten. Diese Eigenschaft wird *Führungsverhalten* des Regelkreises genannt. Die dem Regler zugeführte Regelabweichung e wird aus der Führungsgröße und der Regelgröße wie folgt berechnet:

$$e = r - y \tag{2.1}$$

Des Weiteren sollten Störungen z, die auf die Strecke einwirken umgehend ausgeregelt werden. Ziel ist es, dass auch bei einer stetigen Störungseinwirkung die Regelgröße den Wert der Führungsgröße annimmt. Diese Eigenschaft wird *Störverhalten* des Regelkreises genannt. Die Abbildung 2.3 zeigt das typische Einschwingverhalten der Regelgröße y des Regelkreises auf einen sprunghafte Veränderung der Führungs- r bzw. Störgröße z.

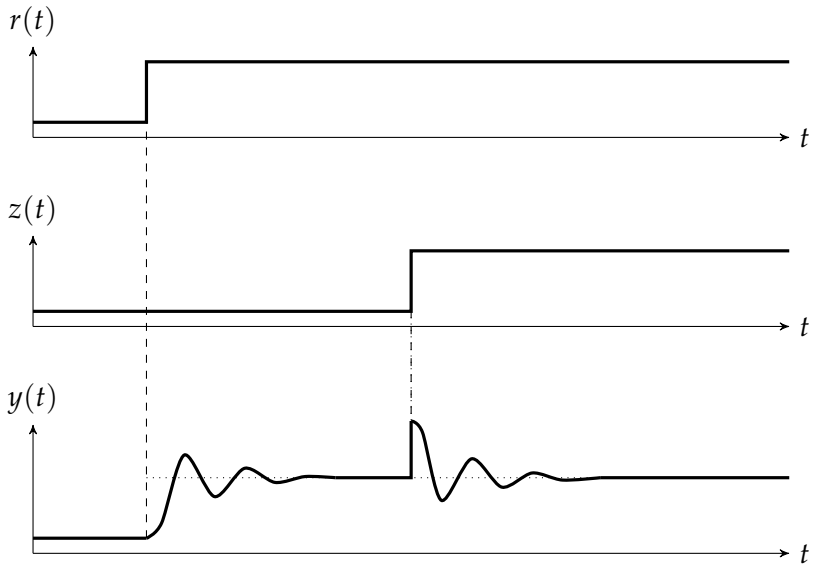


Abbildung 2.3: Führungs- und Störverhalten eines Regelkreises

2.1.2 Reglerstruktur

Je nach geforderter Regelgenauigkeit und dem geforderten Zeitverhalten der Regelstrecke muss eine passende Reglerstruktur gewählt werden. Sie beschreibt die Eigenschaften des Reglers wie z.B. die Typen von Übertragungsgliedern, aus denen ein Regler aufgebaut ist. Die drei unterschiedlichen Anteile eines PID-Reglers können wie in Abbildung 2.4 dargestellt, zusammengesetzt werden. Nachfolgend werden die Reglergrundtypen sowie die wichtigsten zusammengesetzten Übertragungsglieder mit ihren Übertragungsfunktionen aufgeführt:

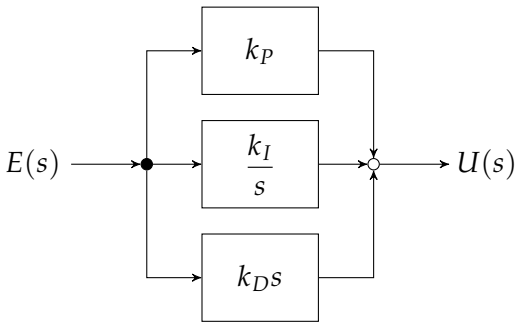


Abbildung 2.4: Struktur eines PID-Reglers [Lun10]

P-Regler

Der P-Regler besteht nur aus einem Proportionalanteil k_P. Dieser verstärkt ($k_P > 1$) bzw. mindert ($k_P < 1$) die Regelabweichung e. Der P-Regler gibt das Ergebnis ohne Verzögerung weiter, wodurch ein schneller Abbau der Regelabweichung möglich ist. Im Allgemeinen kann diese Abweichung aber nicht vollständig abgebaut werden.

$$K_P(s) = k_P \tag{2.2}$$

I-Regler

Der I-Regler verhindert eine bleibende Regelabweichung bei einem sprungförmigen Führungs- und Stellsignal. Die Regelabweichung wird über die Zeit integriert. Solange eine Regelabweichung auftritt ($e \neq 0$), wird die Stellgröße verändert. Die Nachstellzeit T_I beeinflusst dabei den Anstieg. Wird z.B. eine Nachstellzeit von $T_I = 5\,s$ gewählt, benötigt der Regler bei einem konstanten Eingangswert e genau $5\,s$ bis der Ausgang den selben Wert erreicht.

$$K_I(s) = \frac{1}{T_I s} \tag{2.3}$$

PI-Regler

Der PI-Regler kombiniert den P- und den I-Regler. Der Regler hat somit den Vorteil, dass er schnell auf Veränderungen reagieren kann und zudem exakt ausregelt, d.h. keine

bleibende Regelabweichung hinterlässt.

$$K_{PI}(s) = k_P \ (1 + \frac{1}{T_I s}) \tag{2.4}$$

PD-Regler

Der PD-Regler wirkt proportional als auch differentiell auf die Regelabweichung e. Dazu wird der P-Regler mit einem D-Anteil kombiniert. Der D-Anteil bewertet die Veränderung der Regelabweichung und reagiert bei einer starken Veränderung mit einer starken Regelung der Stellgröße. Wie auch der P-Regler hinterlässt der PD-Regler eine bleibende Regelabweichung, reagiert dafür noch schneller auf Veränderungen.

Die Vorhaltzeit T_D beeinflusst dabei die Intensität der Änderung. Sie definiert die Zeit, die ein reines P-Glied früher reagieren müsste, um die Regelabweichung auszugleichen.

$$K_{PD}(s) = k_P \ (1 + T_D s) \tag{2.5}$$

PID-Regler

Der PID-Regler kombiniert die drei Anteile und vereint sämtliche Eigenschaften der P-, I- und D-Anteile. Es bleibt keine bleibende Regelabweichung und es kann sehr schnell auf Veränderungen reagiert werden.

$$K_{PID}(s) = k_P \ (1 + \frac{1}{T_I s} + T_D s) \tag{2.6}$$

Im Optimalfall reagiert ein Regler ohne Verzögerung auf eine Änderung der Führungsgröße und kann auch Störungen ohne Verzögerung ausgleichen. In der Realität benötigt ein Regler eine gewisse Zeit um Veränderungen auszugleichen (Abb. 2.3). Je nach Auswahl der Übertragungsfunktion des Reglers pendelt sich die Regelgröße schneller oder langsamer bzw. mit mehr oder weniger stark schwindenden Verlauf auf den gewünschten Wert ein. Bei realen Problemstellungen muss individuell entschieden werden welches Einschwingverhalten als optimal bezeichnet werden kann.

2.2 Android

Android ist ein auf dem Linux Kernel basierendes, freies Betriebssystem für mobile Endgeräte. Am häufigsten ist es auf Smartphones und Tablets verbreitet, doch auch in der Industrie, unter anderem im Automobil-Bereich, kommt es mehr und mehr zum Einsatz [The13].

Im Jahr 2003 wurde Android von der von Andy Rubin gegründeten Android Inc. entwickelt. Zwei Jahre später wurde das Unternehmen von Google aufgekauft und die Entwicklung von einem Konsortium namens Open Handset Alliance weitergeführt. Neben Google sind unterschiedliche Handy-Hersteller, Halbleiter-Unternehmen, Software-Unternehmen und Mobilfunkanbieter Mitglieder des Konsortiums. Im Oktober 2013 stellte Google die derzeit aktuelle Version 4.0 (alias KitKat) vor (Stand Dezember 2013).

2.2.1 Architektur

Die Architektur des Betriebssystems besteht aus mehreren Schichten. Wie in Abbildung 2.5 zu sehen bildet der Linux Kernel die Basis der Android Architektur. Der Kernel beinhaltet alle nötigen Gerätetreiber für Display, Wi-Fi, Kamera, Speicher, Audio und sonstiger Hardware die angesteuert werden soll. Hinzu kommen die Prozess-, Speicher- und auch die Energieverwaltung. Die weiteren Schichten der Android Architektur werden in folgendem knapp erläutert.

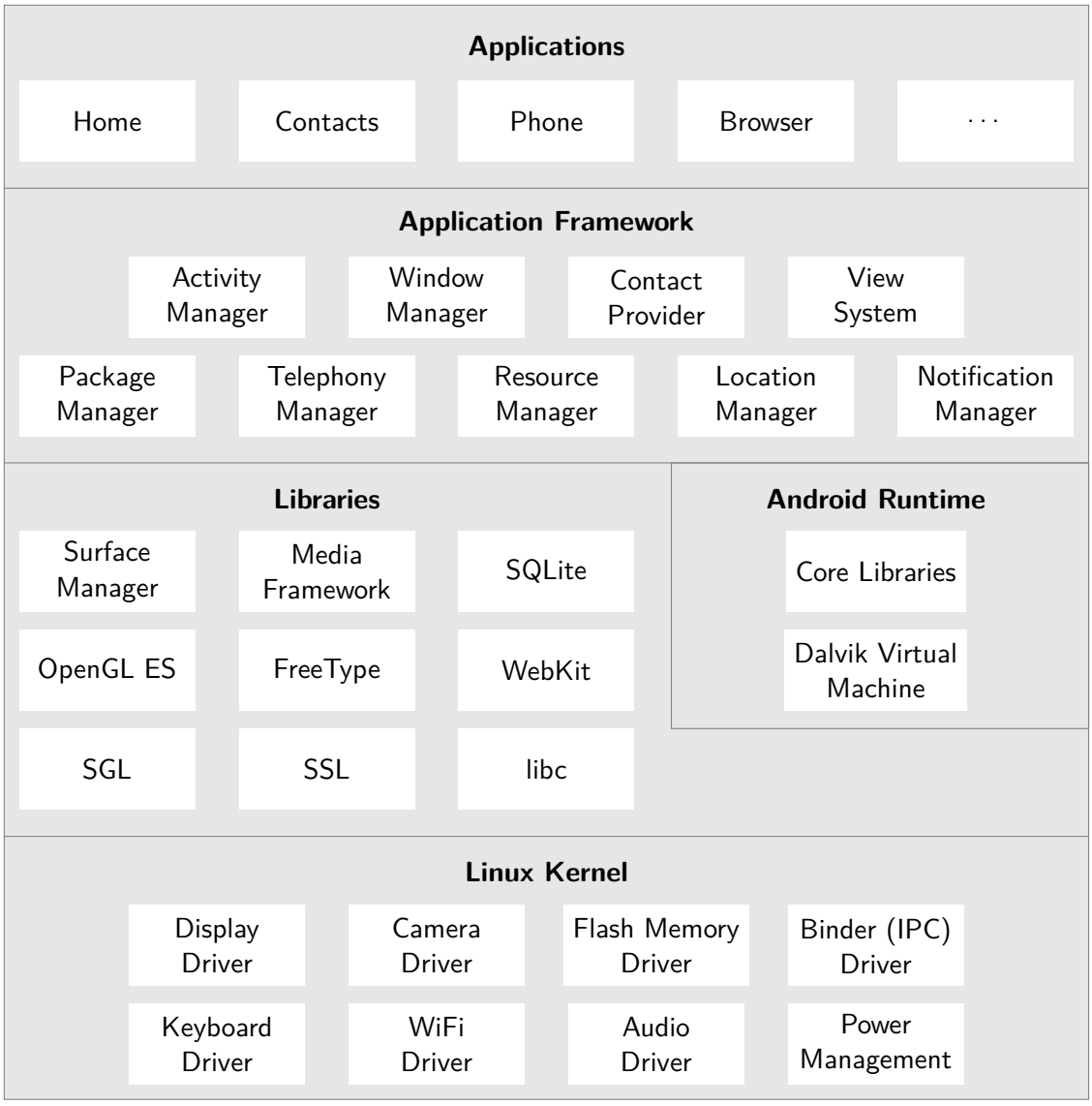


Abbildung 2.5: Schematischer Aufbau der Android-Architektur [Kün12]

Android Laufzeitumgebung (Runtime)

Die Laufzeitumgebung der in Java entwickelten Anwendungen ist die Dalvik Virtual Machine (kurz Dalvik VM). Die VM wurde speziell für den Einsatz in mobilen Endgeräten entwickelt und unterscheidet sich daher von der klassischen Java-Laufzeitumgebung (englisch Java Runtime Environment, kurz JRE). Jede Anwendung die ausgeführt wird startet in einer eigenen virtuellen Maschine. Dies bringt einen erheblichen Vorteil in puncto Sicherheit und Verfügbarkeit, kostet jedoch extra Ressourcen.

Java-Anwendungen können nicht ohne Weiteres auf der Dalvik VM ausgeführt werden. Die kompilierten Java-Klassen (*.class*) müssen mit dem Tool *dx* in Dalvik Executables (*.dex*) umgewandelt werden. Diese Aufgabe wird von der Entwicklungsumgebung automatisch ausgeführt und der Entwickler muss sich darüber keine Gedanken machen.

Bibliotheken (Libraries)

In der Android Architektur kommen unterschiedliche (Standard-)Bibliotheken zum Einsatz. Dazu gehört beispielsweise SQLite, ein schlankes Datenbankmanagementsystem zur Verwaltung von relationalen Daten oder die freie HTML-Rendering-Engine WebKit. Auch die im Spiel verwendete Grafikbibliothek OpenGL ES ist in dieser Schicht zu finden.

Programmierschnittstelle/Anwendungsrahmen (Application Framework)

Sämtliche Klassen für die Verwaltung von Anwendungen, Diensten, Ressourcen oder Sensoren finden man im Anwendungsrahmen. Diese Schicht ist eine einheitliche Schnittstelle für die teils sehr heterogene Hardwarelandschaft, die mittlerweile verfügbar ist. Anwendungsentwickler könne so ohne Rücksicht auf spezielle Hardwareeigenschaften des Endgeräts ihre Anwendungen entwickeln.

Anwendungsschicht (Applications)

In der Anwendungsschicht befinden sich sämtliche Apps, die von Google oder auch von Dritten für Android entwickelt wurden. Auf jedem Smartphone finden man in dieser Schicht Anwendungen zum Telefonieren, Kontakte verwalten, SMS schreiben und viele Weitere. Diese Schicht ist somit für die Interaktion zwischen Mensch und Endgerät zuständig.

2.2.2 Komponenten

Für die Entwicklung von Anwendungen stehen vier zentrale Komponenten zur Verfügung. Es müssen jedoch nicht alle vier Komponenten in einer Anwendung vorhanden sein. Viele Anwendungen verwenden ausschließlich die Activity Komponente. Im Folgenden werden die vier Komponenten kurz beschrieben.

Activity

Über Activties findet die gesamte Interaktion zwischen dem Nutzer und der Anwendungen statt. Aus diesem Grund braucht jede App, die Eingaben vom Nutzer erwartet, mindestens eine Activity. Sie dienen zur Verwaltung von Oberflächen und zur Darstellung. Komplexerer Anwendungen bestehen oft aus einer Vielzahl von Activities zwischen denen ein Nutzer hin und her wechselt. Eine detaillierte Beschreibung zur Verwaltung und zum gesamten Lebenszyklus einer Activity sind in [Her13] von Dominik Herr zu finden.

Service

Teile einer Anwendung die keine Benutzeroberfläche benötigen werden Service genannt. Meist sind dies Aufgaben mit einer hohen Laufzeit, die im Hintergrund ausgeführt werden. Beispielweise soll die Wiedergabe eines Musikplayers auch weiterhin im Hintergrund laufen, auch wenn der Anwender die Benutzeroberfläche der Anwendung bereits verlassen hat. Die Interaktion mit einem Service wird im Allgemeinen über einen Activity realisiert, es besteht aber auch die Möglichkeit einen Service über interaktive Widgets oder Benachrichtigungen zu steuern.

Content Provider

Der Content Provider ist für das Verwalten, Laden und Speichern von Daten zuständig. Des Weiteren kann über Berechtigungen festgelegt werden welche Daten anderen Anwendungen zur Verfügung gestellt werden.

Broadcast Receiver

Ein Broadcast Receiver wird verwendet, um sich in das interne Benachrichtigungssystem von Android einzuklinken. Benachrichtigung können über sogenannte Intentfilter abonniert werden und somit kann eine Anwendung auf Ereignisse des Systems oder anderer Anwendungen reagieren. Wird etwa der Akku des Geräts schwach, sendet das System über das Benachrichtigungssystem die Meldung *ACTION_BATTERY_LOW*. Alle registrierten Anwendungen erhalten diese Benachrichtigung und können beispielsweise mit einer Warnmeldung oder anderen Aktionen auf dieses Ereignis reagieren.

2.2.3 Sensoren

Für die Navigation oder für die Interaktion mit dem Gerät spielen Sensoren eine zentrale Rolle. Um die exakte Position für die Navigation ermitteln zu können wird ein GPS Sensor benötigt. Will man zu der Position noch die Ausrichtung des Geräts wissen, wird zusätzlich ein Magnetfeldsensor benötigt.

Android unterstützt eine Vielzahl von unterschiedlichen Sensoren. Diese können in die drei Kategorien Positionssensor (P), Bewegungssensor (B) und Umgebungssensor (U) eingeteilt werden. Einige der Sensoren sind nur Softwaresensoren (S) deren Werte durch die Kombination von Hardwaresensoren (H) berechnet werden. Beispielweise gibt der lineare

Beschleunigungssensor die Differenz des Beschleunigungs- und Gravitationssensors an. Der Rotationssensor gibt den Achswinkel des Geräts in *rad* an. Berechnet werden diese Werte durch den Gravitations- und Magnetfeldsensor. Dabei muss berücksichtige werden, dass die berechneten Werte relativ zur Erde ausgegeben werden. Details diesbezüglich werden im Kapitel 3.3 beschrieben.

Bei anderen Sensoren, etwa beim Schwerkraftsensor werden, die Werte relativ zum Gerät bereitgestellt. Eine Zusammenstellung aller unterstützen Sensoren ist in Tabelle 2.1 zu finden.

Sensor	Kat.*	Typ**	Beschreibung
TYPE_ACCELEROMETER	H	B	Misst die Beschleunigung (inklusive der Schwerkraft) die auf das Gerät einwirkt in m/s^2.
TYPE_AMBIENT_TEMPERATURE	H	U	Misst die Umgebungstemperatur in Grad Celsius (°C).
TYPE_GRAVITY	H/S	B	Misst die Schwerkraft in m/s^2
TYPE_GYROSCOPE	H	B	Misst die Rotationsgeschwindigkeit in rad/s.
TYPE_LIGHT	H	U	Misst die Umgebungshelligkeit (Beleuchtung) in lux.
TYPE_LINEAR_ACCELERATION	H/S	B	Misst die Beschleunigung (exklusive der Schwerkraft) die auf das Gerät einwirkt in m/s^2.
TYPE_MAGNETIC_FIELD	H	P	Misst das Magnetfeld in μT
TYPE_ORIENTATION	S	P	Misst die Orientierung des Geräts in allen drei physikalischen Achsen. Dieser Sensor ist seit Android 2.2 veraltet (deprecated).
TYPE_PRESSURE	H	U	Misst den Luftdruck in hPa oder $mbar$.
TYPE_PROXIMITY	H	P	Misst den Abstand von Objekten zum Display in cm. Dieser Sensor wird oft verwendet, um zu überprüfen ob das Gerät am Ohr des Nutzers anliegt.
TYPE_RELATIVE_HUMIDITY	H	U	Misst die relative Luftfeuchtigkeit in Prozent (%).
TYPE_ROTATION_VECTOR	H/S	P	Misst die Orientierung des Geräts im Roll-Nick-Gier-Winkel.
TYPE_TEMPERATURE	H	U	Misst die Temperatur des Geräts in °C. Dieser Sensor wurde ab Android 4.0 durch den Sensor TYPE_AMBIENT_TEMPERATURE ersetzt, da er oft von Geräteherstellern falsch interpretiert wurde.

Tabelle 2.1: Überblick der von Android unterstützen Sensortypen [Gooa].
* S: Softwaresensoren, H: Hardwaresensoren
** P: Positionssensor, B: Bewegungssensor, U: Umgebungssensor

3 Konzept

In diesem Abschnitt wird zunächst der Aufbau des Regelungstechnik-Spiels beschrieben und erklärt welche Konzepte hinter der Architektur des Spiels stecken. Im zweiten Teil werden die regelungstechnischen Grundkonzepte, die sich hinter den einzelnen Spielmodi verbergen, beschrieben und abschließend der mathematische Hintergrund der Sensorsteuerung erläutert.

3.1 Architekur

Sehr abstrakt gesehen besteht das gesamte Spiel aus zwei miteinander verknüpften Komponenten. Zum einen aus der grafischen Darstellung (Raumschiff, Trajektorie, Sterne, Head-up-Display) und zum anderen aus dem mathematischen Modell unseres dynamischen Systems (Raumschiff). Um diese beiden Komponenten zu verknüpfen wurde das Entwurfsmuster Model View Controller verwendet. Zunächst wird in folgendem Abschnitt das Paradigma erläutert und anschließend die genaue Architektur des Spiels dargestellt.

3.1.1 Model View Controller

Model View Controller, kurz MVC, ist ein Entwurfsmuster für einen effizienteren und strukturierten Aufbau von Software [ES10]. Im Allgemeinen helfen Entwurfsmuster beim Design und der Architektur in der Softwareentwicklung. Meist stecken erprobte Designentscheidungen hinter einem Muster, die eine wertvolle Unterstützung für Software-Entwickler und Software-Architekten bieten. Obwohl die Verwendung von solchen Entwurfsmustern einen deutlichen Mehraufwand in der Entwurfsphase eines komplexen Softwareprojekts bedeutet, senkt der Einsatz von Mustern das Risiko von Entwurfsfehlern enorm. Dennoch muss darauf geachtet werden, dass der Mehraufwand keine unnötige Komplexität mit sich bringt. Es sollte stets darauf geachtet werden den Entwurf einer Software so einfach wie möglich zu gestalten.

Bei der Entwicklung des Spiels *Space-Controller* wurde versucht sich möglichst exakt an das Entwurfsmuster Model View Controller zu halten. Dabei handelt es sich um ein Paradigma den Code in kleine Funktionseinheiten zu verpacken, um nicht die Übersichtlichkeit zu verlieren. Primär besteht das Entwurfsmuster aus drei Komponenten. Die erste Komponente wird als Model bezeichnet, die zweite Komponente als View. Verbunden werden diese beiden Teile durch den Controller [BMR+96]. Die Aufgaben der einzelnen Komponenten sind dabei folgende:

Model

Das Model repräsentiert die Daten eines Programms. Neben der Verwaltung der Daten müssen auch Änderungen am Zustand vorgenommen werden. Auf Anfrage wird dieser Zustand einer View-Komponente zur Visualisierung bereitgestellt. Dabei kann ein Model nicht nur von einer einzelnen View verwendet werden, sondern von beliebig vielen. Über Änderungen am Zustand werden die registrierten Views informiert und können gegebenenfalls den aktuellen Zustand anfordern.

View

Eine View ist eine Komponente zur Visualisierung von Daten. Sie holt die Daten direkt von verschiedenen Models und stellt diese für den Nutzer dar. Die View ist mit einem Model verbunden und wird von diesem über Änderungen am Zustand informiert. So kann direkt auf Änderung reagiert und der neue Zustand dargestellt werden.

Controller

Der Controller verarbeitet die Eingaben des Nutzers. Die Eingaben, die vom Nutzer auf einer View-Komponente gemacht wurden, werden vom Controller entgegengenommen und auf Funktionen dieser View-Komponente oder eines Models abgebildet. Der Controller ist somit die Schnittstelle für Änderungen am Model. Er hat sowohl Zugriff auf das Model als auch auf die Views, um Anpassungen vorzunehmen.

Nach einer Interaktion mit dem Nutzer ergibt sich ein Ablauf wie in Abbildung 3.1 dargestellt. Nachfolgend werden die einzelnen Schritte die in der Zeichnung abgebildet sind im Detail erläutert.

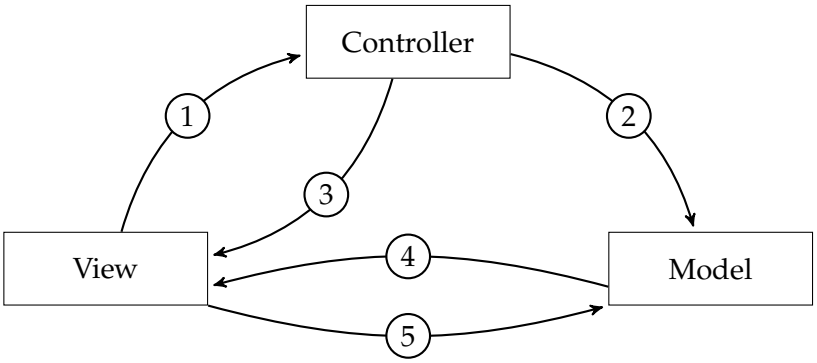


Abbildung 3.1: Interaktionsschema des Model-View-Controller Kozepts

1. **Nach Interaktion des Nutzers mit der View**
   Die View ist für die Darstellung der Daten eines oder mehrerer Models zuständig. Nach einer Interaktion des Nutzers mit der View, beispielsweise durch Klicken auf

eine Schaltfläche, wird diese Änderung dem Controller mitgeteilt. Der Controller ist nun dafür zuständig die Informationen weiter zu verarbeiten.

2. **Änderungen werden im Model verarbeitet**
   Der Controller nimmt die Information entgegen und versucht diese zu verstehen. Die Aufgabe des Controllers ist somit, die Aktion des Nutzers korrekt zu interpretieren und gegebenenfalls die passenden Änderungen am Model vorzunehmen.

3. **Direkte Anpassung der View (Optional)**
   Nachdem eine Controller die Daten verarbeitet hat, können umgehend Änderungen an der View nötig sein. Es könnte zum Beispiel sein, dass einige Buttons deaktiviert andere wiederum aktiviert werden. In diesem Schritt wird jedoch nur auf Änderungen reagiert, die durch die Nutzeraktion entstanden sind.

4. **Model informiert View über Änderungen an den Daten**
   Zwischen Model und View existiert ein sogenanntes publish-subscribe Entwurfsmuster. Dabei werden sämtliche Views, die die Daten eines Models verwenden über Änderungen am Zustand informiert.

5. **View holt aktuelle Daten zur Visualisierung**
   Im letzten Schritt holt sich die View die aktuellen Daten um diese für den Nutzer anzuzeigen. Dieser Prozess kann auf zwei unterschiedliche Arten angestoßen werden. Möglichkeit eins ist die eben erwähnte Änderung der Zustandes der Daten (4), die zweite Möglichkeit ist die in Punkt 3 beschriebene Änderung durch den Controller.

3.1.2 Aufbau

Zu Beginn des Kapitels wurde der Aufbau des Spiels mit zwei abstrakten Komponenten beschrieben. In der Realität ist die gesamte App etwas komplexer da vor Beginn des Spiels noch diverse Einstellungen vorgenommen werden können. So ist es beispielsweise möglich die Reglerparameter anzupassen. Außerdem kann das Verhaltens des Orientierungssensors kann über die Einstellungen angepasst werden. Das Spiel ist in die im Folgenden näher beschriebenen acht Teile aufgesplittet, wobei sich diese Aufteilung ausschließlich auf die grafischen Komponenten des Spiel beschränkt.

Startbildschirm

Das Startbild ist gleichzeitig noch das Hauptmenü des Spiels. Von dort können die Einstellungen, weiterführenden Informationen und die Homepage des Instituts für Systemtheorie und Regelungstechnik aufgerufen werden. Über eine weitere Schaltfläche gelangt man zum Highscore, der Übersicht der bereits erzielten Leistungen im Spiel. Über den zentral angeordneten Play-Button wird man zur Level-Auswahl weitergeleitet.

Einstellungen

In den Einstellungen kann der Spieler seinen Namen festlegen der nach einem erfolgreichen Spiel in die Highscore-Liste eingetragen werden soll. Mit der Option "Beschleunigungsverhalten" kann der Spieler den Orientierungssensor für seinen Bedürfnisse anpassen. Zur Auswahl stehen ein lineares (Standardwert), ein exponentielles oder ein logarithmisches Verhalten des Beschleunigungssensors zur Verfügung. Einzelheiten zum Verhalten des Sensors und zur Implementierung werden in Abschnitten 3.3 und 4.3 beschrieben.

Spielmodus-Auswahl

Die Spielmodus-Auswahl bietet eine Übersicht über die derzeit vier verschiedenen Hauptspielmodi. Mittels einer Wischgeste von rechts nach links oder andersherum kann zwischen den Modi gewechselt werden. Durch einen Klick auf eine der Schaltflächen kann der entsprechende Modus ausgewählt werden.

Menü zur Wahl der Regelparameter

Je nach gewähltem Modus können unterschiedliche Parameter angepasst werden. Die jeweiligen Parameter werden in Abschnitt 3.2 genauer beschrieben. Zusätzlich zu den Parametern kann eine Trajektorie gewählt werden, die im Spiel vom Raumschiff bestmöglich verfolgt werden muss. Um zu überprüfen, ob die gewählten Parameter zum gewünschten Verhalten des Reglers führt, kann bei einigen Spielmodi noch ein passendes Diagramm angezeigt werden.

Diagramme

Für eine einfache Wahl der Reglerparameter stehen zwei unterschiedliche Diagramme zur Verfügung. Wird der Spielmodus "Loop-Shaping" gewählt kann der Amplitudengang und der Phasengang der Übertragungsfunktion in einem Bode-Diagramm angezeigt werden. Bei den Spielmodi "Zustandsrückführung" und "Beobachterentwurf" können die Pole und Nullstellen der Übertragungsfunktion angezeigt werden.

Spieloberfläche

Das eigentliche Spiel wird in der Spieloberfläche dargestellt. Diese besteht aus zwei Teilen, einem OpenGL-Renderer und einem Head-up-Display (HUD). Der OpenGL-Renderer erstellt die grafische Repräsentation des Spiels, welche aus den vier Bausteinen Raumschiff, Trajektorie, Sterne und Hintergrund besteht. Hinzu kommt noch das HUD welches den aktuellen Punktestand und den Vorschritt anzeigt. Eine genauere Beschreibung beider Komponenten wie auch Details zur Implementierung sind in [Her13] von Dominik Herr zu finden.

Zusammenfassungsbildschirm

Wie der Name "Zusammenfassungsbildschirm" schon sagt werden dort nach dem Spiel nochmals sämtliche Parameter und Einstellungen aufgelistet. Dazu gehört der Name des Spielers, die erreichte Punktzahl, die Platzierung im Highscore sowie die gewählten Regler- und Sensorparameter. In einem Schaubild wird zusätzlich zur erreichten Punktzahl der Verlauf der Trajektorie und die Verfolgungskurve des Raumschiffs dargestellt. Weiterführende Informationen sind wiederum in [Her13] zu finden.

Highscore

Der Highscore ist eine Auflistung alles absolvierter Spiele. Hierbei wird sowohl der Name des Spielers als auch die im Spiel erreichte Punktzahl aufgelistet. Um die Punktzahlen miteinander vergleichen zu können, wird für jeden Spielmodus und für jede Trajektorie eine separate Liste geführt.

3.2 Reglerentwurf

Dieser Abschnitt gibt einen Überblick über die im Spiel implementierten Entwurfsverfahren. Über die verschiedenen Spielmodi kann das gewünschte Entwurfsverfahren bestimmt werden. Zusätzlich müssen die Regelparameter so gewählt werden, dass die an den Regelkreis gestellten Anforderungen bestmöglich erfüllt werden. Eine Grundanforderung an einen Regelkreis ist die Stabilität. Ein System gilt als stabil, wenn eine beliebige und beschränkte Änderung des Eingangssignals auch nur eine beschränkte Änderung des Ausgangssignal hervorruft. Dabei kann die beschränkte Änderung entweder als eine Auslenkung $\mathbf{x}_0$ aus der Gleichgewichtslage verstanden werden oder das System wird von Außen durch die Eingangsgröße erregt.

In [Lun10] definiert Jan Lunze die vier Schritte, die unabhängig vom Entwurfsverfahren zur Lösung einer Reglerentwurfsaufgabe führen. Gegeben ist dabei ein Modell der Regelstrecke sowie die Güteanforderungen an den Regelkreis. Ziel ist ein Regelgesetz (Reglerstruktur und Reglerparameter) mit der der Regelkreis die gegebenen Güteanforderungen erfüllt.

1. Überführung der Güteanforderungen in für den gewählten Reglerentwurf passende Ersatzanforderungen
2. Festlegung der Reglerstruktur und der Reglerparameter
3. Analyse des Reglerverhaltens
4. Bewertung der Güte des Regelkreises. Werden mit der gewählten Struktur und den Parametern die Güteanforderungen erfüllt? - Sollte dies nicht der Fall sein wird erneut in Schritt 1 oder 2 gestartet.

Im Fall des "Space-Controller" Spiels ist bereits eine passende Reglerstruktur vorgegeben. Der erste Schritt und der erste Teil des zweiten Schritts sind somit gegeben. Der

Spieler muss sich ausschließlich um die Wahl der geeigneten Reglerparameter kümmern. Für die Wahl der Parameter und die anschließende Analyse stehen je nach Entwurfsart unterschiedliche Diagramme zur Verfügung. Die Güte des Reglers wird durch die erreichte Punktzahl widergespiegelt. Nachfolgend werden die einzelnen Entwurfsverfahren genauer beschrieben.

3.2.1 Zustandsrückführung

Bei der klassischen Regelung, wie in Kapitel 2 beschrieben, wird stets der Ist-Sollwert-Vergleich verwendet. Bei einer Zustandsregelung werden die inneren Zustände x_i des Systems mit einem Faktor k_i gewichten und diese wiederum zurückgeführt. Abbildung 3.2 zeigt den Aufbau eines Zustandsreglers als Blockschaltbild.

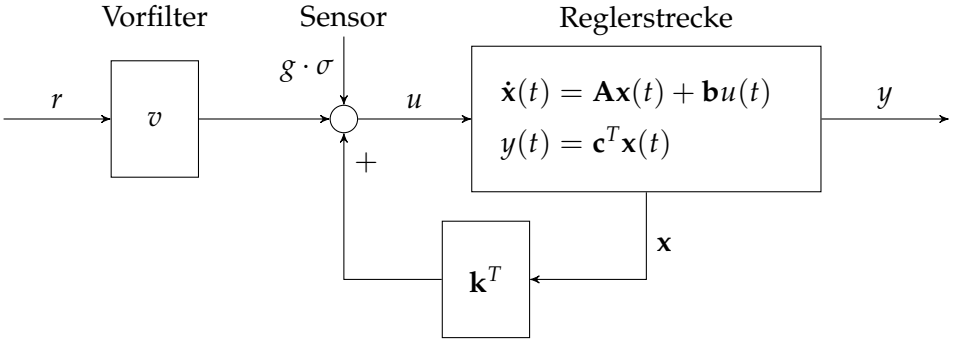


Abbildung 3.2: Blockschaltbild eines Zustandreglers mit zusätzlichem Sensoreingang

Um die Funktionsweise des Zustandsreglers beschreiben zu können, benötigt man zunächst ein Zustandsraummodell des Systems. Das Übertragungsverhalten eines dynamischen Systems kann durch eine Differenzialgleichung n-ter Ordnung beschrieben werden. Das Zustandsraummodell besteht aus einem System von n Differenzialgleichungen erster Ordnung. Die zwei verschiedenen Darstellungen können voneinander abgeleitet werden und sind somit eine mathematisch äquivalente Beschreibung des dynamischen Systems. Durch die einfache Handhabung des Zustandsraummodells wird diese Darstellung in ingenieurtechnischen Bereichen zur Analyse und zum Reglerentwurf bevorzugt eingesetzt. Allgemein wird das Zustandsraummodell in folgende Form dargestellt:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{b}u(t), \quad \mathbf{x}(0) = \mathbf{x}_0 \\ y(t) &= \mathbf{c}^T\mathbf{x}(t) + du(t) \end{aligned} \tag{3.1}$$

$\mathbf{A}$ ist im Allg. eine konstante (n, n)-Matrix, welche auch als **Systemmatrix** bezeichnet wird, $\mathbf{x}$ ein n-dimensionaler Vektor welcher die zeitabhängigen Elemente $x_i(t)$ enthält. Der Eingangsvektor $\mathbf{b}$ ist ein n-dimensionaler Spaltenvektor. In der zweiten Gleichung, der Ausgabegleichung, ist $\mathbf{c}^T$ ein n-dimensionaler Zeilenvektor und d eine Skalar. Der

Durchgangsfaktor d ist bei vielen Systemen gleich null. Die Anfangsbedingungen der einzelnen Elementen x_i werden im n-dimensionalen Vektor $\mathbf{x}_0$ beschrieben, welcher im Allg. bekannt sein muss.

Der Vektor $\mathbf{x}$ wird Zustand eines dynamischen Systems genannt. Er definiert für einen beliebigen Zeitpunkt $t_e \geq 0$ die einzelnen Elemente $x_i(t)$ so, dass zusammen mit dem Verlauf der Eingangsgröße $u(\tau)$ für $0 \leq \tau \leq t_e$ die Werte der Ausgangsgröße $y(t_e)$ eindeutig bestimmt sind. $\mathbf{x}$ wird dabei auch Zustandsvektor genannt und die einzelnen Komponenten $x_i(t)$ von $\mathbf{x}$ werden Zustandsvariablen oder Zustandsgrößen genannt [Lun10]. Sie tragen die Informationen über das dynamisch Verhalten einer Regelstrecke.

Bei einem Zustandsregler werden die einzelnen Zustandsvariablen zurückgeführt. Durch die Rückführung ist es möglich, die Pol- und Nullstellen oder Eigenwerte, welche das Zeitverhalten einer Regelung festlegen, der Regelung vorzugeben. Aus Abbildung 3.2 lässt sich die folgende Gleichung für den Zustandsregler ableiten:

$$u_k(t) = \mathbf{k}^T \cdot \mathbf{x}(t) \; . \tag{3.2}$$

Des Weiteren wird der Vorfilter $\mathbf{v}$ mit der Gleichung

$$u_w(t) = v \cdot w(t) \tag{3.3}$$

beschrieben. Der Vorfilter gleicht eine bleibende Regelabweichung, die durch die Proportional-Elemente des Rückführungsvektors entstehen, wieder aus. Der Vorfilter kann allgemein mit der Gleichung

$$v = \left[\mathbf{c}^T \left(-\mathbf{A} + \mathbf{b}\mathbf{k}^T \right)^{-1} \mathbf{b} \right]^{-1} \tag{3.4}$$

beschrieben werden. Wie in Kapitel 4, zu sehen ist, wird diese Formel jedoch nicht verwendet. Der Spieler muss selbst einen passenden Wert für den Vorfilter berechnen oder diesen experimentell bestimmt.

3.2.2 Zustandsrückführung mit Beobachter

Ein Beobachter wird verwendet wenn die internen Zustandsvariablen eines Systems nicht gemessen werden können. Dazu wird ein mathematisch identisches Modell des dynamischen Systems für den Beobachter gebildet. Das Modell wird als Zustandsgleichung angegeben. Im spiel wird ein sogenannter Luenberger-Beobachter verwendet, der wie in Abbildung 3.3 veranschaulicht, parallel zu dem zu regelnden System geschalten wird. Dabei wirkt die Eingangsgröße $w(t)$ auf beide Systeme.

Im Optimalfall sind beide Systeme identisch, so dass beide zu einem beliebigen Zeitpunkt t den selben Zustandsvektor besitzen: $\mathbf{x}(t) = \hat{\mathbf{x}}(t)$. In der Realität tritt dieser Sonderfall allerdings selten auf. Aus diesem Grund muss das Modell des Beobachters geregelt werden. Dazu wird die Differenz

$$\tilde{y}(t) = y(t) - \hat{y}(t) \tag{3.5}$$

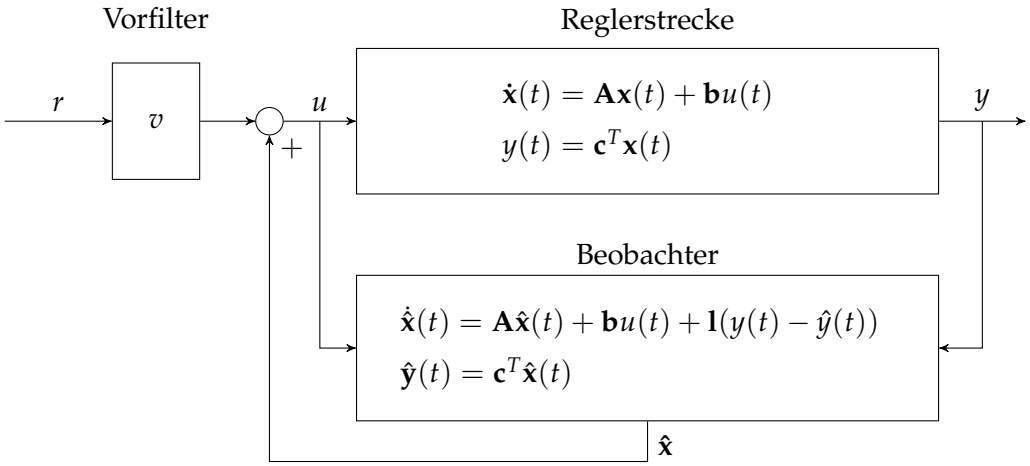


Abbildung 3.3: Blockschaltbild eines Systems mit Beobachter

zwischen dem Ausgang der Regelstrecke und dem Ausgang des Beobachter gebildet. Diese Differenz wird Beobachterfehler genannt und wird für die Reglung des Beobachters ausgenutzt. Dazu werden die Zustandsgrößen jeweils über einen P-Regler an das jeweilige System zurückgeführt. Die einzelnen Elemente l_i von $\mathbf{l}$ können frei gewählt werden. Die einzige Voraussetzung ist die Stabilität des Systems, die mit Hilfe des Pol-Nullstellen-Diagramms überprüft werden kann. Der Regler des Beobachters kann wie folgt beschrieben werden:

$$u_b = \mathbf{l} \cdot (y(t) - \hat{y}(t)) = \mathbf{l} \cdot \tilde{y}(t) \ . \tag{3.6}$$

Daraus ergibt sich für den gesamten Beobachter folgendes Zustandsraummodell:

$$\begin{aligned} \dot{\hat{\mathbf{x}}}(t) &= \mathbf{A}\hat{\mathbf{x}}(t) + \mathbf{b}u(t) + \mathbf{l}(y(t) - \hat{y}(t)) \\ \hat{\mathbf{y}}(t) &= \mathbf{c}^T\hat{\mathbf{x}}(t) \ . \end{aligned} \tag{3.7}$$

3.2.3 Loop Shaping

Ein weiteres Entwurfsverfahren ist das Loop Shaping. Wie der Name des Verfahrens schon sagt wird dabei die Übertragungsfunktion geformt, d.h. gezielt verändert. Die Anpassungen werden bei diesem Verfahren am offenen Regelkreis vorgenommen um damit die gewünschten Eigenschaften des geschlossenen Kreises zu erzielen. Die Anforderungen an den geschlossenen Kreis sind:

- Stabilität
- gutes Führungs-/Störverhalten
- kleine bleibende Regelabweichung

- schnelles Einschwingverhalten
- Robustheit der Stabilität

Dieses Verfahren gehört zu den Frequenzbereichsverfahren und hat den Vorteil, dass die Übertragungsfunktion des offenen Kreises linear von der Übertragungsfunktion des Systems abhängt. Wie die Gleichung (3.9) zeigt, ist der Zusammenhang des geschlossenen Kreises mit dem System nichtlinear. Ob die geforderten Anforderungen an den geschlossenen Kreis erfüllt sind, kann mit Hilfe des Bode-Diagramms oder der Nyquist Ortskurve (siehe Abbildung 3.4) überprüft werden. Ist dies nicht der Fall kann dies durch Anpassen der Reglerparameter erreicht werden. Im Spiel steht ausschließlich das Bode-Diagramm für die Analyse zur Verfügung. Details zur Implementierung sind in Kapitel 4.5 zu finden.

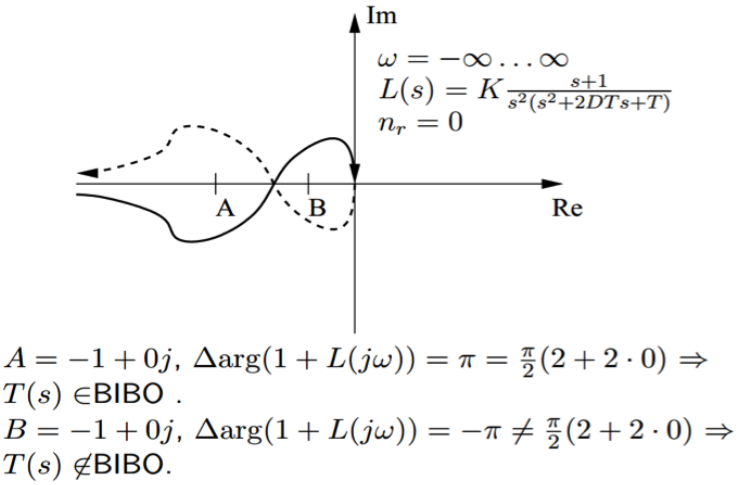


Abbildung 3.4: Nyquist Ortskurve

Die Übertragungsfunktion des dynamischen Systems wird mit $G(s)$ bezeichnet, der zu entwerfende Regler mit $K(s)$. Daraus ergibt sich die Übertragungsfunktion

$$G_0(s) = G(s) \cdot K(s) \tag{3.8}$$

für den offenen Kreis. Für den geschlossenen Kreis erhält man die Gleichung

$$G_w(s) = \frac{GK}{1 + GK} \ . \tag{3.9}$$

Um die gewünschten Eigenschaften für den geschlossenen Kreis zu erzielen, formuliert man die Anforderungen an den Verlauf des Amplituden- und Phasengangs des offenen Kreises. Dies ist im Eingrößenfall einfach, da eine Multiplikation von $G(s)$ und $K(s)$ einer Addition der Amplituden und Phasen $G(s)$ und $K(s)$ im Bode-Diagramm entspricht. Die einzelnen Frequenzen im Bode-Diagramm werden in doppelt logarithmischer Darstellung aufgetragen.

3.3 Sensoren

Im Abschnitt 2.2.3 wurden die unterschiedlichen Sensoren vorgestellt, die unter Android zur Verfügung stehen. Für das Spiel werden der Magnetfeldsensor *TYPE_MAGNETIC_FIELD* und der Beschleunigungssensor *TYPE_ACCELEROMETER* verwendet, um die manuelle Steuerung des Raumschiffs zu realisieren. Dazu werden aus den Werten der beiden Sensoren eine Rotationsmatrix R und eine Neigungsmatrix I berechnet. Im Nachfolgenden wird ausschließlich die Rotationsmatrix R berücksichtigt, da die Neigungsmatrix I keinerlei Relevanz für die entwickelte Anwendung hat.

Die Rotationsmatrix transformiert einen Vektor **a** aus dem Gerätekoordinatensystem in das Weltkoordinatensystem. Das Weltkoordinatensystem ist eine Orthonormalbasis aus den drei Vektoren X, Y und Z.

X ist definiert als das Vektorprodukt aus Y und Z. Er zeigt tangential zum Erdboden in Richtung Osten.

Y ist ebenfalls tangential zum Erdboden an der aktuellen Stelle des Geräts ausgerichtet, zeigt jedoch in Richtung des Nordpols.

Z zeigt in den Himmel und steht exakt senkrecht auf dem Erdboden.

Somit ist R die Einheitsmatrix wenn das Gerät genau nach dem Weltkoordinatensystem ausgerichtet ist. D.h. wenn die x-Achse des Geräts noch Osten, die y-Achse des Geräts zum Nordpol und der Bildschirm Richtung Himmel ausgerichtet ist.

3.3.1 Mathematischer Hintergrund

Die Rotationsmatrix oder auch Drehmatrix beschreibt die Drehung des Gerätekoordinatensystem in Bezug auf das Weltkoordinatensystem. Dabei erfolgt die Drehung stets um eine sogenannte Drehachse. Die Ausrichtung der Richtungsvektoren der Drehachsen legt dabei den Drehwinkel fest. Blickt man in positiver Achsrichtung, so erfolgt eine positive Drehung (Drehwinkel $\phi \geq 0$) im Uhrzeigersinn. Die Drehung um die x-Achse wird durch die Abbildungsgleichung

$$d_1 = x\ \mathbf{e_1} + (y\cos\alpha - z\sin\alpha)\ \mathbf{e_2} + (y\sin\alpha + z\cos\alpha)\ \mathbf{e_3} \tag{3.10}$$

und die Abbildungsmatrix

$$R_X(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix} \tag{3.11}$$

ausgeführt. Für die Drehungen um die y- bzw. z-Achse gibt es die Drehmatrizen

$$R_Y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \tag{3.12}$$

$$R_Z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.13}$$

Um nun einen gegebenen Vektor v z.B. um ϕ Grad um die x-Achse zu drehen, wird dieser mit der Rotationsmatrix R_X multipliziert. Ein positiver Drehwinkel ($\phi \geq 0$) entspricht dabei einer Drehung im Uhrzeigersinn.

$$v' = R_X(\phi)\ v \tag{3.14}$$

Möchte man die Drehung rückgängig machen bzw. eine Rotation gegen den Uhrzeigersinn ausführen, so kann die neue Lage durch Multiplikation mit der inversen Rotationsmatrix bestimmt werden:

$$v' = R_X^{-1}(\phi)\ v\ . \tag{3.15}$$

In [Cra89, S. 45ff.] beschreibt John J. Craig die **Z-Y-X Euler Winkel**-Konvention. Sie beschreibt die Rotation eines Koordinatensystems in Bezug auf ein Referenzsystem. Dabei wird zunächst eine Rotation um die x-Achse des Zielkoordinatensystems mit dem Winkel γ, dann ein Rotation um die y-Achse mit dem Winkel β und zuletzt eine Rotation mit dem Winkel α um die z-Achse ausgeführt. Oft wird diese Konvention als Roll-Pitch-Yaw-Winkel (z. Dt. Roll-Nick-Gier-Winkel) bezeichnet.

3.3.2 Roll-Nick-Gier-Winkel

Die Roll-Nick-Gier-Winkel werden viel in der Fahrzeugtechnik, speziell in der Luftfahrt zur Flugsteuerung verwendet. Sie beschreiben die drei einzelnen Drehungen die benötigt werden um das objektbezogenen Koordinatensystem (Gerätekoordinatensystem) aus dem Weltkoordinatensystem abzuleiten. Die Abbildung 3.5 zeigt anschaulich die Drehung der drei Winkel am Beispiel eines Smartphones.

Der **Roll**-Winkel rotiert das Gerätekoordinatensystem um die x-Achse, der **Pitch**-Winkel rotiert um die y-Achse und der **Yaw**-Winkel definiert die Drehung um die Horizontal-Achse (z-Achse). Die Ausrichtung des Gerätekoordinatensystems kann je nach Hersteller oder Formfaktor des Geräts unterschiedlich definiert sein (siehe Abschnitt 4.3). Zu beachten ist, dass die Rotationen in dieser Darstellung um das Gerätekoordinatensystem ausgeführt werden und nicht um das Weltkoordinatensystem.

Bei der Berechnung der Winkel, bzw. zunächst der Rotationsmatrix $R_{X'\,Y'\,Z'}$ kommt es auf die richtige Reihenfolge der Rotationen an. Dies ich wichtig, da die zweite bzw. dritte Rotation von den vorangegangenen Rotationen abhängen. So erhält man durch korrekte Multiplikation der einzelnen Rotationsmatrizen (3.11), (3.12) und (3.11) die Rotationsmatrix

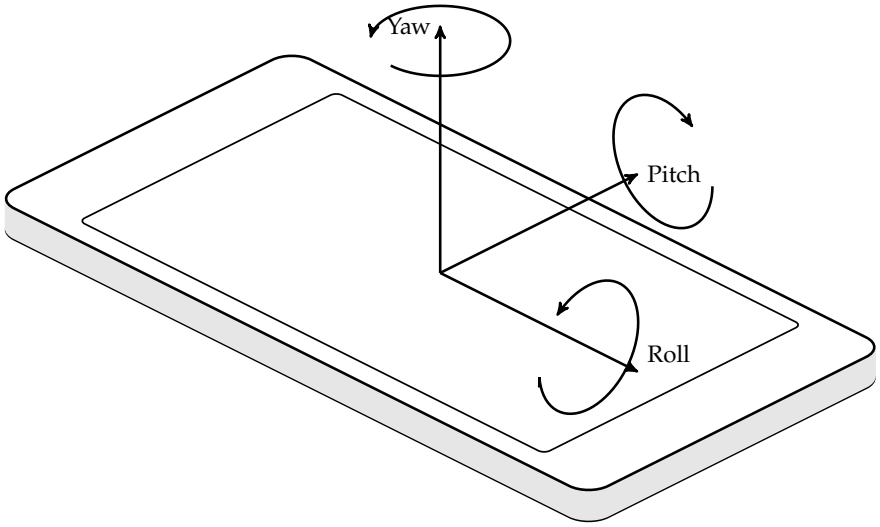


Abbildung 3.5: Veranschaulichung der Roll-Pitch-Yaw Winkel.

$$
\begin{aligned}
R_{X'\,Y'\,Z'} &= R_Z(\alpha)\; R_Y(\beta)\; R_X(\gamma) \\
&= \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix} \\
&= \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}.
\end{aligned} \tag{3.16}
$$

Beim inversen Problem, der Berechnung der Winkel mit gegebener Rotationsmatrix $R_{X'\,Y'\,Z'}$ stehen neun Gleichen zur Verfügung. Jedoch hängen sechs dieser neun Gleichungen voneinander ab, wodurch drei Gleichungen und drei Unbekannte resultieren. Mit einer gegebenen Rotationsmatrix

$$
R_{X'\,Y'\,Z'}\,(\gamma,\beta,\alpha) = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix} \tag{3.17}
$$

können die drei gesuchten Winkel wie folgt berechnet werden.

$$\beta = \text{Atan2}\left(-r_{3,1}, \sqrt{r_{1,1}^2 + r_{2,1}}\right) \tag{3.18}$$

$$\alpha = \text{Atan2}\left(\frac{r_{2,1}}{\cos\beta}, \frac{r_{1,1}}{\cos\beta}\right) \tag{3.19}$$

$$\gamma = \text{Atan2}\left(\frac{r_{3,2}}{\cos\beta}, \frac{r_{3,3}}{\cos\beta}\right) \tag{3.20}$$

Die Funktion Atan2 ist ein Arkustangens mit zwei Argumenten. Diese Funktion wird in vielen Programmiersprachen angeboten und dient zur Umrechnung von kartesischen Koordinaten in Polarkoordinaten. In Java steht die Funktion **Atan**2 in der Klasse Math (java.lang.Math) zur Verfügung und liefert als Resultat den Winkel Θ, der nach der Umwandlung in Polarkoordinaten entsteht.

Probleme treten auf, wenn ein **Roll**-Winkel $\beta = \pm\frac{\pi}{2}$ gewählt wird. Hierbei spricht man von einem Gimbal Lock bzw. einer kardanischen Blockade [Cra89]. Es kommt zu einer Überlagerung der ersten und der dritten Rotation, wodurch nur noch die Summe der beiden Rotationen ausschlaggebend ist. Damit verliert man einen Freiheitsgrad, d.h. eine Rotation um die ursprünglich x-Achse ist nicht mehr möglich.

Der Wertebereich der Winkel ist beim **Yaw**-Winkel von $0\,^\circ$ bis $360\,^\circ$, wobei ein Winkel von $0\,^\circ$ eine Ausrichtung des Geräts nach Norden, 90° eine Ausrichtung nach Osten, 180° eine Ausrichtung nach Süden und $270\,^\circ$ eine Ausrichtung nach Westen bedeutet. Der **Pitch**-Winkel, die Rotation um die y-Achse, hat einen Wertebereich von $-180\,^\circ$ bis $180\,^\circ$, der **Roll**-Winkel einen Bereich von $-90\,^\circ$ bis $90\,^\circ$.

4 Implementierung

Die aktuelle Implementierung des Spiels "Space-Controller" verwendet die Android SDK Version 18. Dies entspricht Version 4.2.*x* alias "Jelly Bean" des Android Betriebssystems, welche im November 2012 erschienen ist [GD12]. Mindestvoraussetzung, um das Spiel ausführen zu können, ist jedoch die Version 2.1 alias Éclair.

In diesem Kapitel wird im Detail auf die Implementierung des Spiels eingegangen. Zunächst wird das Rechtemanagement und die weiteren Aufgaben der Manifest Datei erläutert. Im Anschluss werden die Implementierungen der verschiedenen Regler inklusive derer mathematischen Herleitungen beschrieben. Weiterführend werden die Implementierung des Lagesensors und die Umsetzung der App-Navigation gezeigt. Zum Abschluss werden die in der App zur Verfügung gestellten Diagramme und deren Implementierung vorgestellt.

4.1 App Manifest

Neben grundlegender Informationen wie dem Name des Java Pakets ist die Rechteverwaltung ein essentieller Teil der Manifest Datei. Um beispielsweise das Raumschiff mit dem Lagesensor des Smartphones steuern zu können müssen die folgenden beiden Berechtigungen dem Spiel gewährt werden:

```
1   <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Im Allgemeinen beschränken Berechtigungen den Zugang zu Teilen des Codes oder zu Daten auf dem Gerät. Die Beschränkungen werden auferlegt, um kritische Daten, wie beispielsweise Positionsdaten oder private Nachrichten (SMS), vor Missbrauch zu schützen. Bei der Installation einer App werden die benötigten Rechte gut sichtbar dargestellt und müssen vom Nutzer bestätigt werden. Für das Spiel werden die Berechtigungen *ACCESS_FINE_LOCATION* und *ACCESS_COARSE_LOCATION* benötigt, um auf die Orientierungs- und Positionsdaten des Beschleunigungs- und Magnetfeldsensors zugreifen zu können. Zu Testzwecken ist zusätzlich noch die Berechtigung

```
1   <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

nötig. Diese erlaubt, Daten auf den externen Speicher des Geräts zu schreiben. Verwendet wird die Berechtigung zum Speichern der Sensordaten, um diese später analysieren zu können.

Wie bereits erwähnt, werden neben den Berechtigungen sämtliche grundlegenden Informationen der Anwendung in der AndroidManifest.xml-Datei festgelegt [Goob]. Dazu gehören unter anderem die Festlegung der SDK-Version und der aktuellen Programmversion. Der wichtigste Teil ist der **application** Teil.

Auflistung 4.1: Auszug aus der AndroidManifest.xml-Datei

```
1  <application
     android:icon="@drawable/ic_launcher"
3    android:label="@string/app_name"
     android:allowBackup="true">
5    <activity
       android:name="HomeActivity"
7      android:label="@string/app_name"
       android:screenOrientation="landscape"
9      android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
       <intent-filter>
11       <action android:name="android.intent.action.MAIN"/>
         <category
            android:name="android.intent.category.LAUNCHER"/>
13     </intent-filter>
     </activity>
15
     [...]
17 </application>
```

In diesem Teil, zu sehen in Auflistung 4.1, werden alle Activities, die in der Anwendung verwendet werden, einzeln aufgeführt. In Zeile 6 wird der Name einer Activity festgelegt, der gleichzeitig auch dem Name der Java-Klasse entspricht. Somit kann eine eindeutige Verknüpfung zwischen der Definition in der Manifest-Datei und der Java-Klasse hergestellt werden. Über einen Intent-Filter (siehe Zeile 10ff.) kann festgelegt werden, ob und für welche Activities ein Startsymbol im Launcher angelegt werden soll. Der Launcher bietet in der Regel eine Übersicht aller installierten Anwendungen in Android. Durch Drücken der Haus-Taste gelangt man direkt in den Launcher. Viele Launcher bieten neben der Übersicht der Anwendungen verschieden Bildschirme an, auf denen wiederum Symbole abgelegt werden können.

Ein wichtiger Punkt für das Spiel ist die Ausrichtung des Bildschirms. Da das Spiel mit dem Lagesensor gesteuert wird und daher die Lage des Geräts stets geändert werden muss, ist die automatische Bildschirmausrichtung des Betriebssystems zu deaktivieren. Im Spiel wird die Orientierung bzw. die Ausrichtung der Anzeige jeder Activity mit dem Attribut

```
1 android:screenOrientation="landscape"
```

in der Manifest-Datei im Querformat fixiert. Dies unterbindet ein ungewolltes Drehen des Bildschirms und erleichtert zudem die Implementierung des Sensors.

Da sich diese Arbeit nur mit der Implementierung des Modells (Raumschiff und Regler), dem Sensor, der App-Navigation und der Visualisierung der internen Modelldaten befasst, werden ausschließlich diese Teile nachfolgend betrachtet. Wie bereits in Kapitel 1 erwähnt, ist eine detaillierte Beschreibung der restlichen Komponenten, u.a. der Visualisierung der Spieloberfläche, in [Her13] von Dominik Herr zu finden.

4.2 Regler

Die verschiedenen Regler sind in der „Spaceship"-Klasse (siehe /contoller/Spaceship.java) implementiert. Das implementierte Modell beschreibt ein Raumschiff, welches sich in einem zweidimensionalen kartesischen Koordinatensystem bewegt. Die Berechnung der Position erfolgt mittels numerischer Simulation. Da die Leistung der mobilen Endgeräte es nicht ermöglicht, eine geschlossene Lösung für das Raumschiff zu berechnen, wird auf ein iteratives Lösungsverfahren zurückgegriffen. Um eine Näherungslösung zu finden wird in einem Intervall von $10\,ms$ die neue Position des Raumschiffs berechnet. Das Intervall ist so gewählt, dass bei jeder Neuzeichnung der grafischen Oberfläche neue Positionsdaten vorliegen.

Um nicht die gesamt Anwendung zu blockieren, ist die Berechnung in einen separaten Thread ausgelagert. In jedem Berechnungsschritt wird zunächst die Zeitdifferenz Δt (im Quellcode **dT**) berechnet, die seit der letzten Berechnung vergangen ist. Diese Differenz berechnet sich aus der aktuellen Systemzeit und dem letzten Zeitstempel welcher nach jedem Iterationsschritt gespeichert wird (Auflistung 4.2 Zeile 12).

Auflistung 4.2: Einzelner Berechnungsschritt (ohne Regler)

```
1  float dT = (System.nanoTime() - mTimestamp) / 1000000000.0f;
   // Calculate new X position
3  mPositionX += dT * mSpeedX;
   // Calculate new Y position
5  double r = mTrajectory.getValue(mPositionX) + getNoise();
   double rObs = mTrajectory.getValue(mPositionX);
7  [...]

9  mSpeedY += 0.5 * dT * a;
   mPositionY += dT * mSpeedY;
11 mSpeedY += 0.5 * dT * a;

13 mTimestamp = System.nanoTime();
```

Die Bewegung in X-Richtung ist unabhängig vom Spielmodus und ist konstant. Die konstante Geschwindigkeit v_x ist in **mSpeedX** gespeichert. Die neue Position $x(t)$ welche in **mPositionX** gespeichert ist, wird aus der alten Position $x(t-1)$, der Zeitdifferenz Δt und der Geschwindigkeit v_x wie folgt berechnet:

$$x(t) = x(t-1) + \Delta t \cdot v_x \tag{4.1}$$

Variable (Quelltext)	Variable (Formeln)	Beschreibung
`dt`	Δt	Zeit die zwischen zwei Berechnungsschritten vergangen ist
`r`	r	Sollposition des Raumschiffs (inkl. Messungenauigkeit)
`rObs`	r_B	Sollposition des Raumschiffs für den Beobachter
`mPositionY`	x_1	Y-Position des Raumschiffs
`mSpeedY`	x_2	Geschwindigkeit des Raumschiffs in Y-Richtung
`mSensor.getValue()`	σ	Sensormesswert
`acc`	g	Verstärkungsfaktor des Sensors
`a`	$\dot{x}_2$	Beschleunigung des Raumschiffs in Y-Richtung
`prefilter`	v	Vorfilter der Zustandsrückführung
`k1` und `k2`	k_1 und k_2	Regelparameter der Zustandsrückführung
`l1` und `l2`	l_1 und l_2	Beobachterparameter
`mPositionYObs`	$\hat{x}_1$	Y-Position des Beobachters
`mSpeedYObs`	$\hat{x}_2$	Geschwindigkeit des Beobachters
`aObs`	$\dot{\hat{x}}_2$	Beschleunigung des Beobachters
`kp`, `v` und `w`	k_p, v und w	Loop-Shaping Parameter

Tabelle 4.1: Übersicht aller in Quelltext verwendeten Variablen inklusive der mathematischen Variablen und derer Beschreibung

Die Implementierung ist in Zeile 4 der Auflistung 4.2 zu sehen. Für die Bewegung in Y-Richtung stehen sechs unterschiedliche Modi für die Berechnung der Beschleunigung zur Verfügung. Auflistung 4.3 zeigt einen Ausschnitt aus dem Quellcode mit einer Übersicht der verschiedenen Modi.

Auflistung 4.3: Auflistung der verschiedenen Spielmodi

```
1 public static final int MODE_SENSOR = 0;
  public static final int MODE_FEEDBACK = 1;
3 public static final int MODE_FEEDBACK_HYBRID = 2;
  public static final int MODE_OBSERVER = 3;
5 public static final int MODE_OBSERVER_HYBRID = 4;
  public static final int MODE_LOOP_SHAPING = 5;
```

Der Sollwert der Y-Position wird von der ausgewählten Trajektorie festgelegt. Um das Spiel realistischer zu gestalten, wird in Zeile 7, Auflistung 4.2 auf den Sollwert ein gaußverteilter Messfehler addiert.

Nach der Berechnung der Beschleunigung des Raumschiffs wird die neue Y-Position berechnet. Die Implementierung ist in Auflistung 4.2 Zeile 11-13 zu sehen. Hier wird anstelle eines einfach Euler-vorwärts-Verfahren das Leapfrog-Verfahren für die numerische Integration verwendet. Beim Leapfrog-Verfahren handelt es sich um ein Verfahren zweiter Ordnung wodurch eine genauere Berechnung möglich ist. Die neue Position $x_1(t)$ wird aus der alten Position $x_1(t-1)$, der Zeitdifferenz Δt und der Geschwindigkeit $\dot{x}_1(t-1)$ berechnet. Somit ergibt sich für die Zeilen 11-13 folgenden mathematische Beschreibung:

$$\begin{aligned} x_2(t+0{,}5) &= x_2(t) + 0{,}5 \cdot \Delta t \cdot \dot{x}_2(t) \\ x_1(t+1) &= x(t-1) + \Delta t \cdot x_2(t+0{,}5) \\ x_2(t+1) &= x_2(t+0{,}5) + 0{,}5 \cdot \Delta t \cdot \dot{x}_2(t) \end{aligned} \tag{4.2}$$

Die Berechnung der Beschleunigung $\dot{x}_2(t)$ in den einzelnen Modi und deren Implementierung wird nachfolgend im Einzelnen beschrieben.

4.2.1 Manuelle Stuerung

Im Modus “Manuelle Stuerung” (MODE_SENSOR) wird das Raumschiff rein über die Lage des Geräts gesteuert. Hierzu wird der in Abschnitt 4.3 beschriebene Orientierungssensor verwendet. Über die Auswahltasten (Abb XY) kann die Intensität **acc** des Sensors beeinflusst werden. Die errechnete Beschleunigung a wird, wie in Auflistung 4.2 Zeile 11ff. zu sehen, für die Berechnung der neuen Y-Position verwendet. Abbildung 4.1 zeigt das dynamische System als Blockschaltbild.

Daraus kann die folgende Zustandsraumdarstellung 4.3 für das System abgeleitet werden:

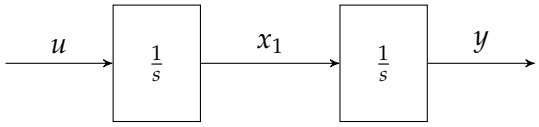


Abbildung 4.1: Blockschaltbild des dynamischen Systems

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ y &= \begin{bmatrix} 0 & 1 \end{bmatrix} \mathbf{x} \end{aligned} \tag{4.3}$$

In Auflistung 4.4 wird die Beschleunigung **a** aus dem frei wählbaren Intensitätsfaktor **acc** und dem aktuellen Messwert des Sensors berechnet. Der Wertebereich des Sensors liegt dabei zwischen -1 und 1. Die Berechnung dieses Werts wird im Abschnitt 4.3 im Detail erläutert.

Auflistung 4.4: Berechung der Beschleunigung für MODE_SENSOR

```
1 a = acc * mSensor.getValue();
```

Für diesen rein gesteuerten Modus berechnet sich der Eingang $u(t)$ des Systems wie folgt aus dem Sensormesswert σ und seinem Verstärkungsfaktor g:

$$u(t) = g \cdot \sigma(t) \tag{4.4}$$

4.2.2 Zustandsrückführung

Im Modus "Zustandsrückführung" (MODE_FEEDBACK) wird die in Kapitel 3 beschriebene Zustandsrückführung verwendet. Aus den Gleichungen (3.2) und (3.3) ergeben sich zusammen mit dem System (4.3) folgende Übertragungsfunktion für die Zustandsrückführung.

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{b}u \\ &= \mathbf{A}\mathbf{x} + \mathbf{b}\left(u_w + u_k\right) \\ &= \mathbf{A}\mathbf{x} + \mathbf{b}\left(\mathbf{k}^T\mathbf{x} + vw\right) \\ &= \left(\mathbf{A} - \mathbf{b}\mathbf{k}^T\right)\mathbf{x} + \mathbf{b}vw \end{aligned} \tag{4.5}$$

Dabei ist **A** die Systemmatrix und **b** der Eingangsvektor. Im konkreten Fall für das Spiel ergibt sich daraus folgende Gleichung:

$$\begin{aligned}\begin{bmatrix}\dot{x_1}\\ x_2\end{bmatrix} &= \left(\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} - \begin{bmatrix}0\\ 1\end{bmatrix}\begin{bmatrix}k_1 & k_2\end{bmatrix}\right)\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix} vr \\ &= \begin{bmatrix}0 & 1\\ -k_1 & -k_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix} vr\,.\end{aligned} \tag{4.6}$$

Aus Gleichung (4.6) kann direkt die Beschleunigung des Raumschiffs abgelesen werden. So erhält man mit den Zustandsvariablen x_1 und x_2 und dem gewünschten Eingangswert r folgende Implementierung (Auflistung 4.5) für die Zustandsrückführung:

Auflistung 4.5: Berechung der Beschleunigung für MODE_FEEDBACK

```
1 a = -k1 * mPositionY - k2 * mSpeedY + prefilter * r;
```

Mathematisch berechnet sich die Beschleunigung wie folgt:

$$\dot{x}_2 = -k_1 x_1 - k_2 x_2 + vr \tag{4.7}$$

Wie in Kapitel 3.2 bereits erwähnt, ist es in der Regel möglich den Vorfilter für ein System mit der Gleichung (3.4) zu berechnen. Da das dynamische System des Raumschiffs allerdings Nullstellen bei $s = 0$ besitzt, kann kein exakter Vorfilter berechnet werden. Somit wird die Berechnung in diesem und auch den folgenden beiden Modi dem Spieler überlassen. Meist ist ein experimentell bestimmter Wert für den Vorfilter schnell gefunden.

4.2.3 Zustandsrückführung mit Sensorsteuerung

Der Modus "Zustandsrückführung mit Sensorsteuerung" (MODE_FEEDBACK_HYBRID) vereint die beiden zuvor beschriebenen Modi. Somit kann noch besser getestet werden, ob der entworfene Zustandsregler die geforderten Eigenschaften erfüllt. Der Spieler kann durch eine abrupte Bewegung des Geräts das Raumschiff von der Trajektorie ablenken und damit testen wie gut das Führungsverhalten des entworfenen Reglers ist. Wie in Abbildung 3.2 zu sehen, werden zur Berechnung der Vertikalbeschleunigung die beiden Einzelbeschleunigungen der vorigen Modi addiert. Man erhält für den Modus MODE_FEEDBACK_HYBRID folgende Implementierung (Auflistung 4.6).

Auflistung 4.6: Berechung der Beschleunigung für MODE_FEEDBACK_HYBRID

```
1 a = -k1 * mPositionY - k2 * mSpeedY + prefilter * r
      + acc * mSensor.getValue();
```

In diesem Modus können die Parameter g, k_1, k_2 und der Vorfilter v frei gewählt werden. Daraus ergibt sich folgende mathematische Gleichung für diesen Modus:

$$\dot{x}_2 = -k_1 x_1 - k_2 x_2 + vr + g\sigma \tag{4.8}$$

Zur Überprüfung der Stabilität und der Robustheit des Reglers steht in den beiden Modi MODE_FEEDBACK und MODE_FEEDBACK_HYBRID ein Pol-Nullstellen-Diagramm zur Verfügung. Details zur Implementierung des Diagramms sind in Abschnitt 4.5.2 zu finden.

4.2.4 Beobachterentwurf

Im "Beobachterentwurf" (MODE_OBSERVER) Modus wird der im Modus MODE_FEEDBACK entworfene Zustandsregler um einen Luenberger-Beobachter mit den Parametern l_1 und l_2 erweitert. Wie in Kapitel 3.2 bereits beschrieben, ist zusätzlich zum eigentlichen Modell ein weiteres Modell für den Beobachter implementiert. Wie in Zeile 1 der Auflistung 4.7 zu sehen, werden die internen Zustände des Beobachtersystems für die Berechnung der Beschleunigung verwendet. Es wird angenommen, dass die internen Zustände des Modells des Raumschiffs nicht gemessen werden können. Des Weiteren wird der Beobachter nicht durch einen Messfehler beeinflusst, wodurch eine bessere Regelung des gesamten Systems möglich ist.

Auflistung 4.7: Berechung der Beschleunigung für MODE_OBSERVER

```
1 a = -k1 * mPositionYObs - k2 * mSpeedYObs + prefilter * r;
2 aObs = -k1 * mPositionYObs - k2 * mSpeedYObs + prefilter *
     rObs
      + l2 * (mPositionY - mPositionYObs);
4
  mSpeedYObs += 0.5 * dT * aObs;
6 mPositionYObs += dT
      * (mSpeedYObs + l1 * (mPositionY - mPositionYObs));
8 mSpeedYObs += 0.5 * dT * aObs;
```

4.2.5 Loop-Shaping

Im "Loop-Shaping" (MODE_LOOP_SHAPING) Modus wird zur Regelung eine Kombination aus einem P-Regler und einem Lead-Element verwendet. Ein Lead-Element ist ein PDT_1-Glied, das ohne die Amplitude zu beeinflussen, die Phase in einem bestimmten Frequenzbereich anhebt. Durch Kombination der beiden Glieder erhält man für das Regelgesetz $K(s)$ die Gleichung

$$K(s) = k_P\, v \cdot \frac{s + w}{s + vw} \qquad v > 1\,. \tag{4.9}$$

Zu beachten ist, dass in diesem Modus v ein Parameter des Lead-Elements und nicht wie in den Modi zuvor der Vorfilter des Eingangssignals r ist. Zusammen mit der Übertragungsfunktion

$$G(s) = \frac{1}{s^2} \tag{4.10}$$

ergibt sich für die Übertragungsfunktion für den offenen Kreis, die Gleichung

$$\begin{aligned} G_0(s) &= K(s) \cdot G(s) \\ &= k_P\, v \cdot \frac{s+w}{s+vw} \cdot \frac{1}{s^2}\,. \end{aligned} \tag{4.11}$$

Durch Anwendung der Gleichung (3.9) folgt für den geschlossenen Regelkreis $G(s)$ des gesamten System die Gleichung

$$\begin{aligned} G(s) &= \frac{G_0(s)}{1+G_0(s)} \\ &= \frac{k_p v s + k_p v w}{s^3 + v w s^2 + k_p v s + k_p v w} \end{aligned} \tag{4.12}$$

Durch geschickte Umformung des Regelgesetzes $K(s)$ lasst sich die Übertragungsfunktion des Reglers in Zustandsraumdarstellung umformen. Für eine einfache Umwandlung muss die Anzahl der Pole größer als die Anzahl der Nullstellen sein. Durch in Umformungen in (4.13) kann dies Voraussetzung erreicht werden.

$$\begin{aligned} K(s) &= \frac{k_p v s + k_p v w}{s+vw} \\ &= \frac{k_p v w - k_p v^2 w + k_p v s + k_p v^2 w}{s+vw} \\ &= \frac{k_p v w(1-v) + k_p v(s+vw)}{s+vw} \\ &= \frac{k_p v w(1-v)}{s+vw} + kv \end{aligned} \tag{4.13}$$

In dieser Form kann die Übertragungsfunktion durch einsetzen der Koeffizienten in Regelungsnormalform (4.14) überführt werden.

$$\begin{aligned} \dot{x} &= \begin{bmatrix} -vw \end{bmatrix} x + \begin{bmatrix} 1 \end{bmatrix} u \\ y &= \begin{bmatrix} kvw\,(1-v) \end{bmatrix} x + kv \cdot u \end{aligned} \tag{4.14}$$

Wie in Auflistung 4.8 in Zeile 1 zu sehen wird zunächst das Glied $K(s)$ integriert. Dazu wird die eben berechnete Zustandsraumdarstellung (4.14) verwendet. Gemeinsam mit dem System 4.3 und der Integration von $K(s)$ ergibt sich für die Beschleunigung in diesem Modus folgende Implementierung (Auflistung 4.8):

Auflistung 4.8: Berechung der Beschleunigung für MODE_LOOP_SHAPING

```
x += dT * ((-v * w * x) + (r - mPositionY));
a = (kp * v * w * (1 - v)) * x + kp * v * (r - mPositionY);
```

4.2.6 Schnittstellen zwischen Modell und Visualisierung

Für die Kommunikation mit dem Raumschiff stehen verschiedene Methoden zur Verfügung. Die wichtigste Methode

```
public double[] getPosition()
```

gibt die aktuelle Position des Raumschiffs aus. Als Rückgabewert erhält man, in einem zweidimensionalen Array, die X- und Y-Position des Raumschiffs. Neben der Position ist auch die Geschwindigkeit des Raumschiffs für die Visualisierung wichtig. Die aktuelle Geschwindigkeit in Y-Richtung beeinflusst die Orientierung des Raumschiffs.

Die Methode

```
public double[] getSpeed()
```

gibt die aktuelle Geschwindigkeit des Raumschiffs aus. Auch in diesem Fall wird ein zweidimensionales Array von der Methode zurückgegeben. Dieses beinhaltet die Geschwindigkeit in X-Richtung, welche allerdings konstant ist, und die Geschwindigkeit in Y-Richtung.

Weitere Methoden geben den aktuellen Punktestand

```
public int getScore()
```

und die Soll-Position des Raumschiffes

```
public double getTrajectoryValue(double x)
```

zurück.

Für die Auswertung nach einem Spiel gibt die Methode

```
public Vector<double[]> getPerformanceObserver()
```

eine Aufzeichnung der Flugkurve des Raumschiffes zurück. Es werden sämtliche Daten, die für die Visualisierung benötigt werden in dieser Klasse gespeichert. Im Entwurfskonzept MVC beschreibt das Raumschiff somit das Modell. Der OpenGL-Renderer, welcher als Controller fungiert, verwendet diese Daten für die Erstellung der grafischen Oberfläche.

4.3 Sensor

Wie bereits in Kapitel 3 beschrieben werden für die manuelle Steuerung des Raumschiffs die Orientierungssensoren (Magnetfeldsensor und Beschleunigungssensor) des Gerätes verwendet. Nach diversen Berechnungen kann die Orientierung des Gerätes im sogenannten Roll-Nick-Gier-Winkel angegeben werden. Für eine bessere User Experience (UX) werden weitere Maßnahmen, darunter die Filterung der Rohdaten und die Definition

einer maximalen und minimalen Auslenkung, unternommen. Die Funktionsweise des Sensors und alle Berechnungen werden im Anschluss anhand von Ausschnitten aus dem Quellcode erläutert.

Der Sensor implementiert das vom Android SDK zur Verfügung gestellte Interface *SensorEventListener*. Nachdem der Listener im *SensorManager*, welcher alle Sensoren verwaltet, registriert ist, wird die Methode

```
public abstract void onSensorChanged (SensorEvent event)
```

bei jeder Änderung eines Sensorwerts aufgerufen. Der übergebende Parameter *event* enthält dabei Informationen über den Sensor, einen Zeitstempel des Ereignisses, die aktuelle Genauigkeit des Sensors und selbstverständlich den gemessenen Sensorwert. Die Genauigkeit des Messwerts wird dabei in drei Stufen (niedrig, mittel und hoch) eingeteilt. Veränderungen der Genauigkeit eines Sensors werden über die Funktion

```
public abstract void onAccuracyChanged (
  Sensor sensor, int accuracy)
```

bereitgestellt, welche ebenfalls durch das Interface *SensorEventListener* implementiert wird.

Der SensorManager berechnet mit Hilfe der Funktion

```
public static boolean getRotationMatrix (
  float[] R, float[] I, float[] gravity, float[] geomagnetic)
```

, wie in Abschnitt 3.3 beschrieben, aus den Messwerten der beiden Sensoren die Rotationsmatrix R und die Drehmatrix I. Da nicht alle Hersteller das gerätegebundene Koordinatensystem gleich definieren, muss die Rotationsmatrix in manchen Fällen gedreht werden. Meist wird das Koordinatensystem anhand der "natürlichen" Ausrichtung des Geräts definiert. D.h. bei Smartphones und kleinen Tablets ist die "natürliche" Ausrichtung im Hochformat, bei größeren Tablets (z.B. Google Nexus 10) ist die "natürliche" Ausrichtung im Querformat. Der SensorManager stellt für die Neuausrichtung des Koordinatensystem eine statische Methode bereit. Auflistung 4.9 zeigt die im Spiel verwendete Neuausrichtung des Koordinatensystems. Die verwendete "natürliche" Ausrichtung (**mRotation**) wird dabei vom System bereitgestellt.

Auflistung 4.9: Neuausrichtung des Koordinatensystems

```
// Default display rotation is portrait
if(mRotation == Surface.ROTATION_0) {
  SensorManager.remapCoordinateSystem(
      R, SensorManager.AXIS_MINUS_Y,
      SensorManager.AXIS_X, R2);
// Default display rotation is landscape
} else {
  SensorManager.remapCoordinateSystem(
      R, SensorManager.AXIS_MINUS_X,
      SensorManager.AXIS_Y, R2);
}
```

Da das Spiel eine fixierte Ausrichtung des Bildschirms hat, müssen nur Hoch- und Querformat betrachtet werden. Die beiden inversen Formate können in diesem Fall ignorieren werden.

Wie bereits in Abschnitt 3.3 erläutert können aus der Rotationsmatrix R mit Hilfe der Formeln (3.20) die einzelnen Drehwinkel berechnet werden. Das Android SDK bietet für diese Berechnung die Methode

```
1 public static float[] getOrientation (float[] R, float[]
    values)
```

an. Diese wird als statische Methode vom SensorManager bereitgestellt. In Auflistung 4.10 ist in Zeile 1 die Implementierung im Spiel zu sehen.

Auflistung 4.10: Koordinaten auslesen

```
1  SensorManager.getOrientation(R2, orientation);
2
3  // orientation contains:
   // [0]:azimut, [1]:pitch and [2]:roll in rad
5  // RAD_TO_DEG = 180.0f / (float)Math.PI
   pitch = orientation[1] * RAD_TO_DEG;
7  roll = orientation[2] * RAD_TO_DEG;
8
9  [...]
10
11 if (Math.abs(pitch) < 90) {
       mOverhead = false;
13 } else {
       mOverhead = true;
15 }
16
17 // Shift array
   for (int i = mValues.length - 2; i >= 0; i--) {
19     mValues[i+1] = mValues[i];
   }
21
   mValues[0] = roll;
```

Die zu berechnenden Werte werden nach Aufruf der Methode in der Variablen **orientation** gespeichert. Für die weiteren Berechnungen sind nur der Pitch- und der Roll-Winkel relevant. Beide Winkel werden in Radian (rad) ausgegeben, für weitere Berechnungen in Grad (°) umgewandelt (Auflistung 4.10, Zeile 6f.). Um das Spielen in jeder Lage (auch Überkopf) zu ermöglichen, wird mittels Pitch-Winkel überprüft, ob sich das Gerät über Kopf befindet oder nicht. Sehr anschaulich ist das in Abbildung 3.5 zu sehen. Wenn der Betrag des Pitch-Winkels einen größeren Wert als 90°erreicht, befindet sich das Gerät in der Überkopflage (**mOverhead** = *true*, Zeile 14).

In den Zeilen 17ff. wird der berechnete Roll-Winkel verarbeitet. Für eine spätere Filterung der Rohdaten werden die letzten zwanzig Sensorwerte in einem Array zwischengespeichert. Dazu werden bei jeder neuen Messung alle alten Werte um eine Position nach hinten verschoben und der neue Wert vorne angefügt. Die Verwendung einer verketteten Liste könnte eine schnellere Ausführung ermöglichen.

Wie eben erwähnt werden die Rohdaten gefiltert, um eventuelle Messfehler zu entfernen. Hierzu wird ein Medianfilter verwendet. Dieser bietet den Vorteil einzeln auftretende Fehler zu ersetzten, ohne dabei eine Glättung der Rohdaten zu bewirken. Eine Glättung der Daten, wie z.B. bei einem Mittelwertfilter, würde ein schwammiges Spielgefühl zur Folge haben, worunter die User Experience leiden würde. Auflistung 4.11 zeigt die Implementierung des Medianfilters im Spiel.

Auflistung 4.11: Medianfilter

```
1 // smooth the values with an median filter
2 float[] values = mValues.clone();
  Arrays.sort(values);
4 float d = values[Math.round((values.length - 1) / 2)];
```

Die Robustheit des Medianfilters gegenüber Ausreißern ist ein weiterer Vorteil gegenüber einem Mittelwertfilter. Treten beispielsweise einzelne Fehler in den Messwerten auf, werden diese ignoriert und haben keine Folgen für die weitere Verarbeitung. Deutlich macht dieses Phänomen die Abbildung 4.2, welche die Rohdaten (blau) und die gefilterten Daten (rot) zeigt.

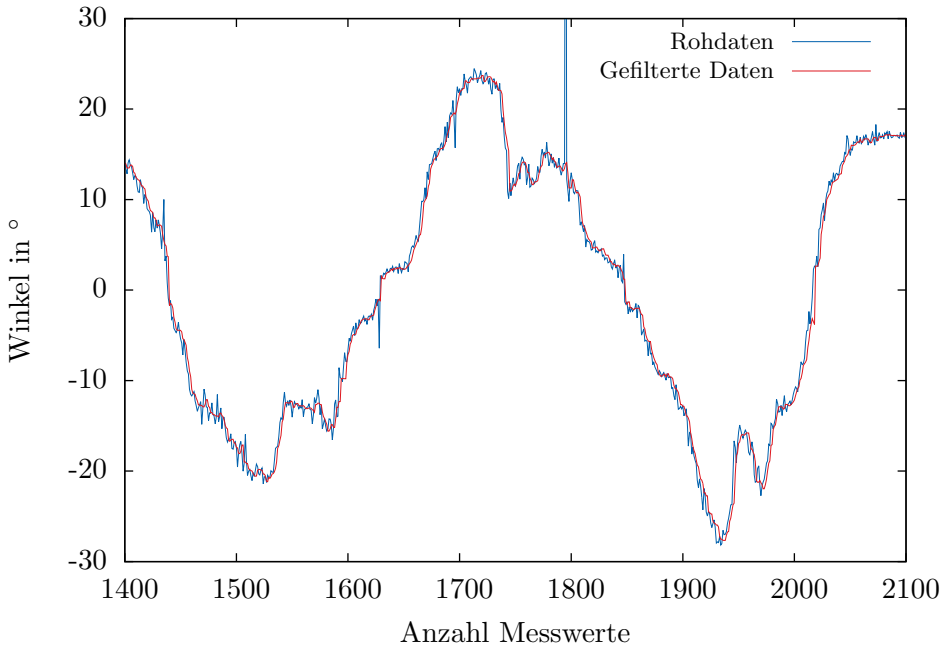


Abbildung 4.2: Vergleich der Rohdaten mit den Daten nach der Filterung

Für eine noch bessere User Experience werden die gefilterten Daten weiter verarbeitet. Zu Beginn eines neuen Spiels wird die aktuelle Position des Geräts festgestellt und gespeichert. Dieser Messwert wird als Referenzwert für die spätere Winkeländerung des Geräts verwendet. Somit kann der Nutzer sein Gerät in einer natürlichen Haltung, was einem Roll-Winkel von ca. $20 - 40\,^\circ$ entspricht, halten, ohne das Raumschiff nach unten zu beschleunigen.

Um das Raumschiff nicht ständig nach oben oder unter zu beschleunigen, wurde bei null Grad ein kleines Plateau eingebaut. In der Auflistung 4.12 in Zeile 8 wird überprüft, ob der Betrag der aktuellen Auslenkung vom zu Beginn gemessenen Referenzwert größer als die Minimalauslenkung ist. Sollte dies nicht der Fall sein, wird eine Auslenkung von null zurückgegeben. Zu beachten ist, dass sich die Werte die zurückgegeben werden in einem Intervall von -1 bis 1 aufhalten. Der maximale bzw. minimale Wert wird schon bei einer Auslenkung von $20\,^\circ$ zurückgegeben.

Auflistung 4.12: Winkel berechnen

```
1  float d = getDegree() - mCalibration;
2
   if (d > 180)
4      d = -180 + (d - 180);
   if (d < -180)
6      d = 180 - (d + 180);

8  if (Math.abs(d) <= MIN_DEFLECTION)
       return 0.0f;
10 else if (d > 0)
       d = d - MIN_DEFLECTION;
12 else
       d = d + MIN_DEFLECTION;
14
   float value = d / (MAX_DEGREE + MIN_DEFLECTION);
16 if (value > 1)
       value = 1;
18 if (value < -1)
       value = -1;
20
   return value;
```

Durch all diese Maßnahmen wird die User Experience deutlich verbessert. Fehler in den Messwerten, die oft bei günstigeren Geräten auftreten, werden in den meisten Fällen vollständig korrigiert. Dies verhindert unerwartete Bewegungen des Raumschiffs, welche den Spieler irritieren.

Mit der Version 3.x des Betriebssystems hat sich einiges an der Implementierung des Orientierungssensor verändert. Wie in Tabelle 2.1 zu sehen, gab es bis zu Version 2.2 einen speziellen Sensor namens **TYPE_ORIENTATION**. In der zweiten Spalte der Tabelle ist abzulesen, dass es sich bei diesem Sensor um einen reinen Softwaresensor handelt. Die Implementierung dieses Sensors ist ähnlich zu der im Spiel verwendeten

Implementierung. Es fehlt allerdings die Neuausrichtung der Rotationsmatrix, wodurch bei vielen Tablets ein fehlerhaften Verhalten auftritt. Durch die Entfernung des Orientierungssensors ist jeder Entwickler selbst für die richtige Ausrichtung der Rotationsmatrix verantwortlich und ist somit deutlich flexibler.

4.4 App-Navigation

Nach dem Start des Spiels über das Raumschiff-Icon erscheint direkt das Hauptmenü (Abbildung 4.3). Vor dort gelangt man über Schaltflächen an den Ecken zu den Einstellungen (4), zum Highscore (1), zur Homepage des Instituts für Systemtheorie und Regelungstechnik (2) und in der letzten Ecke zu weiterführenden Informationen (3) über das Spiel. Über das zentral angeordnete Play-Symbol gelangt man zur Übersicht der Spielmodi.

Abbildung 4.3: Hauptmenü des Spiels

Für die Gestaltung von Oberflächen gibt es bei Android eine eigens dafür entwickelte XML Sprache. Es ist durchaus möglich die gesamte Oberfläche im Quellcode aus einzelnen Komponenten zusammenzubauen. Dies ist jedoch unübersichtlich und nicht so effizient wie eine deklarative Implementierung im XML-Format. Die einzelnen Elemente werden wie bei HTML[1] ineinander verschachtelt. Über Attribute können zusätzlichen Eigenschaften, bezüglich Verhalten und Optik festgelegt werden.

Die einzelnen Komponenten, auch *Views* genannt, können in Gruppen, den sogenannten *ViewGroups*, zusammengefasst werden. In der Auflistung 4.13 werden beispielsweise eine Text-Komponente (**TextView**) und eine Schaltfläche (**Button**) in einer Gruppen zusammengefasst. In diesem Fall handelt es sich um ein **LinearLayout**, welches sämtliche Kindelemente entweder nebeneinander oder wie in diesem Fall untereinander anordnet. Die Ausrichtung des LinearLayouts kann über das Attribut `android:orientation="vertical"` festgelegt werden.

[1] Hypertext Markup Language

Auflistung 4.13: Beispiel eines LinearLayout [Gooc]

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:orientation="vertical" >
  <TextView android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="I am a TextView" />
  <Button android:id="@+id/button"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="I am a Button" />
</LinearLayout>
```

Für die Übersicht der Spielmodi, welche in Abbildung 4.4 zu sehen ist, wurde eine eigene View-Komponente implementiert. Die Komponente besteht aus einem HorizontalScrollView und einem Gestendetektor. Zum erstellten Layout werden die Schaltflächen der einzelnen Spielmodi hinzugefügt und so angeordnet, dass eine Schaltfläche zentral auf dem Display erscheint und die nächste Schaltfläche sich bereits zur Hälfte außerhalb des Bildschirms befindet. Durch eine langsame Wischgeste nach links kann zum nächsten Modus gewechselt werden.

Abbildung 4.4: Übersicht der Spielmodi

Für eine gute User Experience ist es wichtig, dass sich das Layout gleichmäßig mit der Geste bewegt. Beim Nutzer muss das Gefühl entstehen er bewege tatsächliche das Layout mit seinem Finger. Die horizontale Scroll-Funktion ist, wie der Name schon sagt, bereits in der HorizontalScrollView implementiert. Verlässt der Finger des Nutzers den

Touchscreen soll die Schaltfläche des ausgewählten Modus zentral auf dem Bildschirm ausgerichtet werden. Um diese Anforderung umzusetzen wurde das *onTouch*-Event des HorizontalScrollView, wie in Auflistung 4.14 zu sehen, erweitert.

Auflistung 4.14: onTouch Event der HorizontalScrollView

```
1  public boolean onTouch(View v, MotionEvent event) {
2
     //If the user swipes
4    if (mGestureDetector.onTouchEvent(event)) {
       return true;
6    }
     else if (event.getAction() == MotionEvent.ACTION_UP ||
8             event.getAction() == MotionEvent.ACTION_CANCEL ) {
       int scrollX = getScrollX();
10     int width = v.getMeasuredWidth() / 2;
       mCurrent = ((scrollX + (width / 2)) / width);
12     int scrollTo = mCurrent * width;
       smoothScrollTo(scrollTo, 0);
14     return true;
     } else {
16     return false;
     }
18 }
```

Hebt der Nutzer seinen Finger vom Bildschirm wird das *MotionEvent.ACTION_UP*-Event (Zeile 7) ausgelöst. In der darauffolgenden Zeilen wird die gewünschte Position der View berechnet und mit der Methode `smoothScrollTo(scrollTo , 0);` an diese Position gescrollt.

Eine weitere Anforderung an die Komponente ist das Wechseln des aktuellen Modus durch eine kurze und schnelle Wischgeste. Android bietet mit dem *GestureDetector* schon eine Klasse mit den gewünschten Grundfunktionen an. Diese kann mit Hilfe eines *SimpleOnGestureListener* exakt auf die eigenen Bedürfnisse angepasst werden. Durch überschreiben der Methode

```
1  public boolean onFling(MotionEvent e1, MotionEvent e2,
     float velocityX, float velocityY)}
```

kann zusammen mit den Events e_1 und e_2 und der Geschwindigkeit der Fingerbewegung in X-Richtung (*velocityX*) die passende Scroll-Bewegung ausgeführt werden. Ähnlich wie bei onTouch()-Event wird auch hier die Methode `smoothScrollTo(scrollTo , 0);` verwendet.

Die Schaltflächen für die einzelnen Modi sind wiederum in einer Layout-Datei im XML-Format deklariert. Im Layout kann für jede Schaltfläche ein Klick-Event definiert werden. Dazu wird der Methodenname im Attribut `android:onClick` angegeben und bei einem Klick-Event automatisch in der entsprechenden Activity aufgerufen. Die aufzurufende Methode muss vorhanden und mit dem Schlüsselwort *public* gekennzeichnet sein. Andernfalls kommt es bei einem Klick-Event zum Absturz der Anwendung.

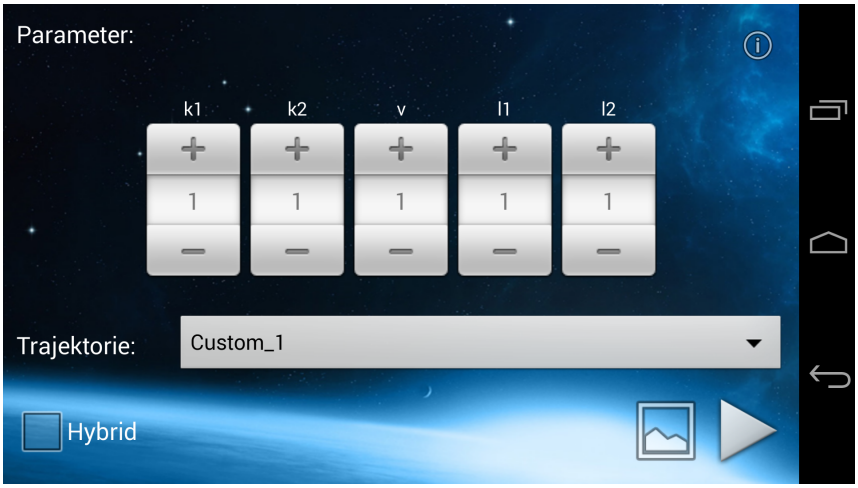


Abbildung 4.5: In der LevelActivity können die Parameter für den gewählten Spielmodus eingegeben werden.

Über die Schaltflächen der einzelnen Modi kommt man zu deren Einstellungen (Abbildung 4.5). Hier können die Parameter des Sensors und der Regler angepasst werden. Bei der Implementierung der Oberfläche (LevelActivity) wurde nach dem DRY-Prinzip (Don't repeat yourself) [ES10, Seite 5ff.] vorgegangen und so eine möglichst generische Lösung zum Einstellen der Parameter erstellt. Der ausgewählte Modus wird beim Start der LevelActivity übergeben. So werden für jeden Modus nur die benötigten Parameter eingeblendet. Die eingestellten Parameter können anhand von Diagrammen vor dem Spiel überprüft werden. Die genaue Implementierung ist in Abschnitt 4.5 zu finden. Über das Play-Symbol in der LevelActivity wird das Spiel mit den zuvor gewählten Parametern gestartet.

4.5 Diagramme

Zur Kontrolle der Parameter stehen je nach Spielmodus die passenden Diagramme 4.5.1 und 4.5.2 zur Evaluierung der Parameter zur Verfügung. In den folgenden beiden Abschnitten wird die genaue Implementierung der beiden Diagramme erläutert. Beim Pol-Nullstellen Diagramm gibt es zusätzlich zur Visualisierung die Möglichkeit mit den Diagramm zu interagieren, um so den Regler exakt dimensionieren zu können.

4.5.1 Bode Diagramm

Das Bode Diagramm, auch Frequenzkennliniendiagramm genannt, besteht aus zwei Kennlinien die in Abhängigkeit zur Kreisfrequenz ω dargestellt werden. Die eine Kennlinie beschreibt den Betrag des Frequenzgangs G, die andere die Phase. Die Kennlinien werden auch *Amplitudengang* und *Phasengang* genannt.

In der Praxis sind oft Frequenzen in einem Bereich von mehreren Zehnerpotenzen interessant, so dass die Skala der Kreisfrequenz ω in logarithmischem Maßstab aufgetragen

wird. Auch der Betrag des Amplitudengangs ändert sich um mehrere Größenordnungen und wird daher ebenso logarithmisch dargestellt. Dieser wird in Dezibel (dB) aufgetragen, wobei sich der in Dezibel angegebene Betrag $|G|_{dB}$ wie folgt aus dem Betrag des Frequenzgangs $|G|$ berechnet.

$$|G(\omega)|_{dB} = 20 \cdot \log_{10} |G(\omega)| \,. \tag{4.15}$$

Die Kennlinie des Amplitudengangs stellt somit die logarithmische Abhängigkeit zum Logarithmus der Kreisfrequenz ω dar. Die Phasenkennlinie stellt die Phase ϕ in Abhängigkeit zur logarithmischen Kreisfrequenz ω dar. Die Ortskurve (Abbildung 3.4) kann einfach durch Kombination der beiden Kennlinien erstellt werden.

Im Spiel wird das Diagramm auf eine Canvas Komponente gezeichnet. Zuvor werden für beide Kennlinien Stützwerte berechnet. Diese werden später als Punkte für die jeweilige Polylinie verwendet. Für die lineare Abbildung der logarithmischen Skala wird zunächst für jeden Stützwert eine passenden Frequenz innerhalb eines vordefinierten Bereichs berechnet. Auflistung 4.15 zeigt diese Berechnung in einem Bereich zwischen den Frequenzen **begin** und **end**.

Auflistung 4.15: Berechnung der äquidistanten Skala

```
double[] values = new double[STEPS + 1];
double delta = (Math.log10(end) - Math.log10(begin)) / STEPS;
for(int i = 0; i <= STEPS; i++) {
    values[i] = Math.pow(10, Math.log10(begin) + (i *
       delta));
}
```

Für jeden Stützwert ω_i wird die passende Amplitude a_i mit

$$\begin{aligned} A &= \frac{k \cdot v}{v\omega_i^2 + m^2} \sqrt{\left(vm^2 + \omega_i^2\right) + \left(vm\omega_i - m\omega_i\right)^2} \\ a_i &= 20 \log_{10} \left(\frac{A}{\omega_i^2} \right) \end{aligned} \tag{4.16}$$

und die Phase ϕ_i mit

$$\phi_i = \arctan \left(\frac{(v-1)\, m\omega_i}{vm^2 + \omega_i^2} \right) - \pi \tag{4.17}$$

berechnet. Nach der Berechnung werden die Punkte verbundenen und man erhält ein Diagramm wie in Abbildung 4.6 zu sehen.

4.5.2 Pol-Nullstellen Diagramm

Die Pole und Nullstellen einer Übertragungsfunktion können mit Hilfe des Pol-Nullstellen Diagramms dargestellt werden. Anhand der Anordnung der Pole und Nullstellen auf der komplexen Zahlenebene des PN-Diagramms können Aussagen über die Stabilität

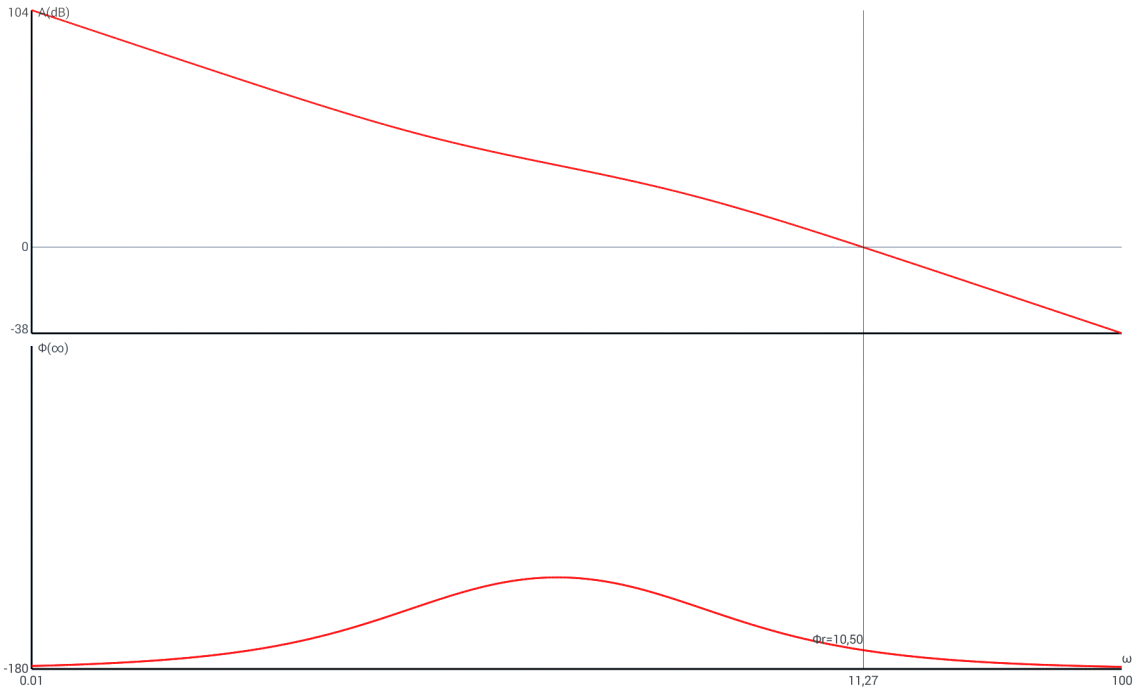


Abbildung 4.6: Bode-Diagramm (Farben invertiert)

eines dynamischen Systems gemacht werden. Befinden sich alle Pole einer Übertragungsfunktion in der linken offenen Halbebene des Diagramms, so ist das System stabil. Abbildung 4.7 zeigt, wie anhand der Anordnung der Pole auf die Stabilität eines Systems geschlossen werden kann. Auch die Kausalität eines Systems kann abgelesen werden, da nur Systeme realisierbar sind bei denen die Anzahl der Pole mindesten so groß wie die Anzahl der Nullstellen ist.

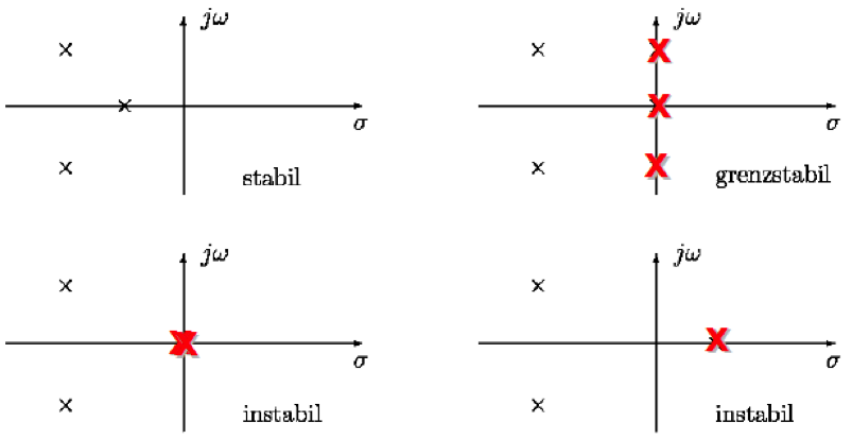


Abbildung 4.7: Stabilitätskriterium in PN-Diagramm

Die Darstellung des Diagramms im Spiel erfolgt wie beim Bode Diagramm auf einer Canvas Komponente. Hauptunterschied ist, dass die Position der dargestellten Pole interaktiv angepasst werden kann. Die Eigenschaft, dass die Pole entweder reell sind oder als komplex konjugiertes Paar auftreten, bringt weitere Einschränkungen, die bei der Implementierung berücksichtigt werden müssen, mit sich. Wie Abbildung 4.8 zeigt, werden die Pole als größere Punkte dargestellt. Dadurch können die Pole leichter mit dem Finger bewegt werden. Mit einer Multitouch-Zoom-Geste kann der Wertebereich des Diagramms angepasst werden. Das Android SDK stellt hierfür den *ScaleGestureDetector* bereit. Durch Überschreiben der Methode

```
1 public boolean onScale(ScaleGestureDetector detector)
```

ist im Spiel ein minimaler und maximaler Zoom-Faktor festgelegt.

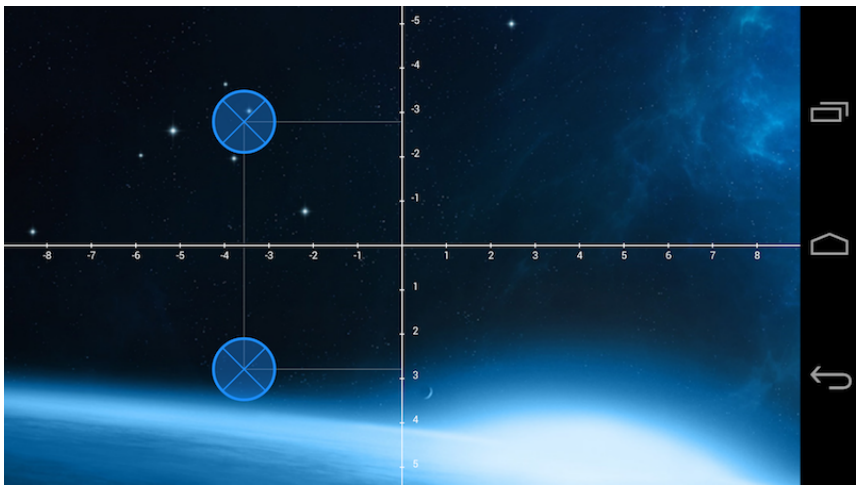


Abbildung 4.8: Pol-Nullstellen-Diagramm

Die Pole können einfach mit dem Finger auf dem Bildschirm verschoben werden. Nähert man sich mit einem Pol der Realachse des Diagramms rasten die Pole ab einem geringen Abstand auf dieser Achse ein. Befinden sich die Punkte nicht auf dieser Achse wird der zweite Pol automatisch an die komplex konjugierte Position gesetzt. Wird ein Pol auf der Realachse abgelegt, befinden sich beide Pole an derselben Stelle. Mit dem Finger kann nun einer der beiden Pole auf der Realachse verschoben werden. Entfernt man sich mit diesem Pol von der Realachse, so wird ein mattierter Pol-Indikator an der konjugiert komplexen Position eingeblendet. Sobald der Nutzer seinen Finger vom Bildschirm entfernt, springt der zweite Pole an die Position des Pol-Indikators. Abbildung 4.9 zeigt die vier Zustände die das Pol-Nullstellen Diagramm annehmen kann.

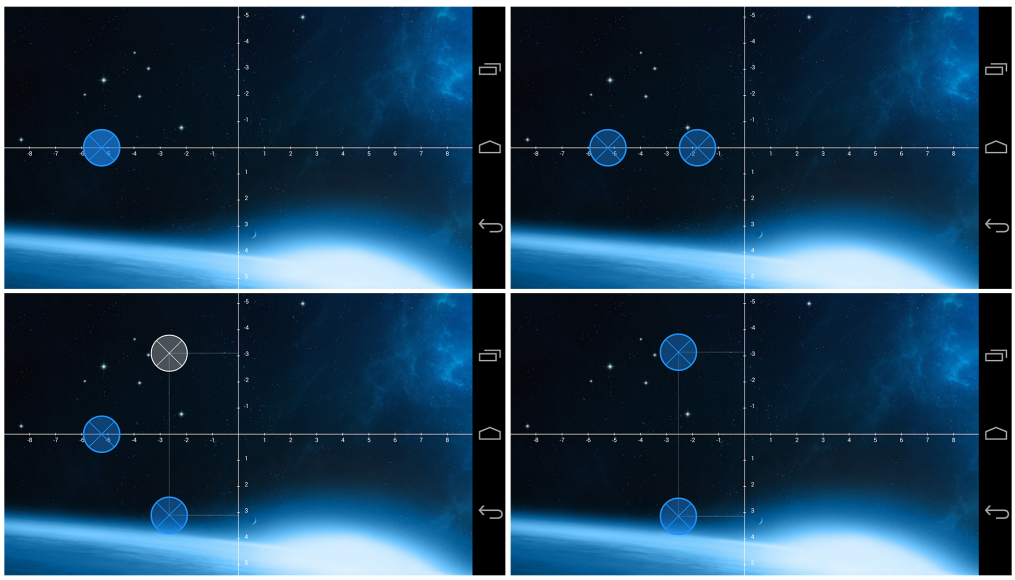


Abbildung 4.9: Verschiedenen Zustände des Pol-Nullstellen-Diagramms

5 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, das E-Learning Angebot für Studierende der Regelungstechnik zu erweitern. Mit dem entwickelten Spiel für Android-Smartphones und -Tablets können die Lehrinhalte spielerisch zugänglich gemacht werden. Mit den verschiedenen Modi werden die in den Vorlesungen behandelten Reglerkonzepte abgedeckt. Wie der nachfolgenden Zusammenfassung zu entnehmen ist, liegt der Schwerpunkt dieser Arbeit auf der Umsetzung der regelungstechnischen Modelle.

Nach einer kurzen Einleitung wurde in dieser Studienarbeit zunächst in die Grundlagen der Regelungstechnik eingeführt und daraufhin einen kurzen Überblick über das Betriebssystem Android gegeben. Hierbei wurde speziell auf die Unterschiede einer Regelung und einer Steuerung eingegangen und die Funktionsweise eines Regelkreises erklärt. Des Weiteren wurde die wichtigsten Reglertypen und deren Eigenschaften aufgezeigt. In Abschnitt 2.2 wurde zunächst die Architektur des Betriebssystems skizziert und anschließend die wichtigsten Komponenten die für die App-Entwicklung benötigt werden beschrieben. Mit einer kurzen Übersicht über die in Android verfügbaren Sensoren 2.2.3 wurde das Grundlagenkapitel abgeschlossen.

Im darauffolgenden Kapitel wurden die Konzepte, die hinter der Implementierung des Spiels stecken, im Detail erläutert. Zunächst wurde das grundlegende Entwurfskonzept "Model View Controller" vorgestellt und der Aufbau des Spiels beschrieben. In Abschnitt 3.2 wurden die drei verschiedenen Reglerentwurfsverfahren vorgestellt die im Spiel später implementiert wurden. Abschließend wurde der mathematische Hintergrund der Verarbeitung der Sensordaten genauer beschrieben.

Im abschließenden Kapitel wurde im Detail auf die Implementierung der App eingegangen. Zu Beginn wurde die Verwaltung der Systemrechte und der Aufbau der Mainfest.xml-Datei beschrieben. Anschließend wurde die einzelnen Implementierungen der Regler vorgestellt und mit deren mathematischen Herleitungen eine Verknüpfungen zu der im Kapitel 3 vorgestellten Konzepte hergestellt. Weiterführend wurde auf die Implementierung des Sensors eingegangen. Dabei wurden verschiedenste Verfahren zur Aufbereitung der Datei vorgestellt. Abschließend wurden die App-Navigation und die verschiedenen Diagramme beschrieben und diverse Aspekte deren Implementierung skizziert.

Alle Anforderungen die an das Spiel gestellt wurden sind ohne bekannten Mängel umgesetzt. Somit sind sämtliche Spielmodi unter Android 2.x oder neuer ohne Fehler spielbar. Es gibt eine Vielzahl von Möglichkeiten die Applikation zu erweitern. Für den manuellen Spielmodus (Steuerung mittels Lagesensors) könnten zukünftig noch weitere Levels mit beweglichen und festen Hindernissen eingebaut werden. Zum besseren Vergleich der einzelnen Spieler untereinander, wäre ein Online-Highscore wünschenswert.

Literaturverzeichnis

[BMR+96] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-oriented Software Architecture: A System of Patterns*. New York, NY, USA : John Wiley & Sons, Inc., 1996. – ISBN 0–471–95869–7

[Cra89] Craig, John J.: *Introduction to robotics: mechanics and control*. 2. Aufl. Boston, MA, USA : Addison-Wesley, 1989. – XIII, 450 S.. – ISBN 0–201–09528–9

[ES10] Eilebrecht, Karl ; Starke, Gernot: *Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung*. 3. Aufl. Heidelberg : Spektrum Akademischer Verl., 2010. – VII, 184 S.. – ISBN 978–3–8274–2525–6

[GD12] Ghosh, Angana ; Dougherty, Dirk: *Introducing Android 4.2, A New and Improved Jelly Bean*. http://android-developers.blogspot.de/2012/11/introducing-android-42-new-and-improved.html. Version: 2012, Abruf: Sonntag, 28. Januar 2014

[Gooa] Google Inc.: *Android API Guides - Sensors Overview*. http://developer.android.com/guide/topics/sensors/sensors_overview.html, Abruf: Sonntag, 28. Januar 2014

[Goob] Google Inc.: *App Manifest | Android Developer*. http://developer.android.com/guide/topics/manifest/manifest-intro.html, Abruf: Sonntag, 28. Januar 2014

[Gooc] Google Inc.: *UI Overview | Android Developer*. http://developer.android.com/guide/topics/ui/overview.html, Abruf: Sonntag, 28. Januar 2014

[Her13] Herr, Dominik: Entwicklung der OpenGL-basierten grafischen Oberfläche eines E-learning-Spiels mit Fokus Regelungstechnik. In: *Institut für Systemtheorie und Regelungstechnik, Universität Stuttgart (unveröffentlicht)* (2013)

[Kün12] Künneth, Thomas: *Android 4: Apps entwickeln mit dem Android SDK*. 2. Aufl. Bonn : Galileo Press, 2012

[Lun10] Lunze, Jan: *Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen: mit ... 75 Beispielen, 165 Übungsaufgaben sowie einer Einführung in das Programmsystem MATLAB*. 8. Aufl. Berlin : Springer, 2010. – XXVII, 710 S.. – ISBN 978–3–642–13807–2

[The13] The Wall Street Journal: *Google, Apple Forge Auto Ties - WSJ.com*. http://online.wsj.com/news/articles/

SB10001424052702304591604579288670734733740. Version: 2013, Abruf: Sonntag, 28. Januar 2014

[Unb08] Unbehauen, Heinz: *Regelungstechnik I: Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*. Berlin : Vieweg+Teubner Verlag, 2008. – ISBN 9783834804976

Eidesstattliche Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum	Unterschrift