PROJEKTBERICHT

Im Studiengang Mechatronik/Robotik

MMR3-17: Die FHTW-Drohne - Programmierung

Ausgeführt von: David Spulak, BSc
Personenkennzahl: 1310331029

BegutachterIn: Dipl.-Ing. Dr. Markus Trenker

Wien, den 22. Januar 2015

FACHHOCHSCHULE
TECHNIKUM WIEN

Kurzfassung

Auf Grund ihrer Stabilität und Manövrierfähigkeit sind Multicopter an der Spitze der Drohnentechnologie. Sie finden einen breiten Einsatzbereich in vielen verschiedenen Industrien, wie der Film- und Spielwarenindustrie aber auch in der Überwachungs- und Militärtechnik. Eine Multicopter-Drohne kann Studenten der FH-Technikum Wien eine herausragende Möglichkeit geben mit einem durch und durch mechatronischen System zu arbeiten und Eröffnet auch den Einstieg in das noch sehr junge Forschungsfeld der Schwarmrobotik.

Die meisten individuellen elektronischen Teile, die zum Erstellen einer Multicopter-Drohne notwendig sind, sind kommerziell erhältlich. Das Know-How zum Bau einer solchen Drohne ist zwar in der Modellbaugemeinschaft vorhanden, für Neulinge und Amateure in diesem Bereich jedoch nur schwer zugänglich. Um diese Wissensdiskrepanz zu schließen wurde im Zuge des Projektes, in dem diese Arbeit entstanden ist, eine Quadcopter-Drohne erstellt, die mit Hilfe einer Smartphone-Applikation gesteuert werden kann. In dieser Arbeit wird die Systemarchitektur der Drohne und die Programmierung der einzelnen Teile vorgestellt, die eine Drohnensteuerung über eine Handyapplikation zulassen. Gezeigt wurde die Einsetzbarkeit dieser Systemarchitektur an einem Prototypen der FHTW-Drohne, der in Laborbedingungen erfolgreich getestet werden konnte.

Schlagworte: Drohne, Quadcopter, Smartphone-Applikation, Steuerung

Inhaltsverzeichnis

1 Einleitung

Unter einer Drohne versteht man im allgemeinen ein unbemanntes Luftfahrzeug, das entweder autonom, semiautonom oder teleoperiert gesteuert wird. In der heutigen Welt finden Drohnen in einer Vielzahl von Bereichen bereits ein Einsatzgebiet. Die derzeit bekanntesten sind wohl die

- Militärtechnik,
- Filmtechnik und
- Spielwarenindustrie.

Im Ganzen stellt eine Flugdrohne eine Kombination aus einer Vielfalt an technischen Disziplinen dar, die Mechanik, Elektrotechnik, Steuerungstechnik und Informatik zu einem durch und durch mechatronischen Gerät vereint. In diesem Sinne ist eine Flugdrohne nicht nur ein valides Mittel Studenten des Studienganges Mechatronik/Robotik das enge Zusammenspiel verschiedener, technisch herausfordernder Gebiete näher zu bringen, sondern auch ein Prestigeobjekt mit dem der Studiengang selbst vermarktet werden kann.

In folgendem Projekt wurden erste Schritt in Richtung eines FH Technikum Wien Drohnen Bausatz gesetzt, der es schlussendlich ermöglichen soll die Drohnentechnologie einer breiten Masse an Studierenden zugänglich zu machen. Dem großen und stetig steigende Anteil an Smartphonebesitzern soll es ermöglicht werden, eine kostengünstige Drohne mit Hilfe ihres Telefons über Bluetooth zu steuern. Zusätzlich kann neben dem Werbeeffekt der für die FH durch eine solche Entwicklung und das damit verbundene Know-How entsteht auch das Forschungsgebiet der FH erweitert werden. Schwarmrobotik ist ein vielseitiges Forschungsfeld, welches, gerade im Gebiet der Flugdrohnen, noch großen Neuheitscharakter aufweist. Gerade ein kostengünstiges Fluggerät ist ein wichtiger Grundstein für jegliche Forschungs- und Projektarbeit in dieser Richtung.

Im Zuge des Projektes, innerhalb dessen diese Arbeit entstanden ist, wurde ein Prototyp für eine Mini-Drohne entwickelt (im Folgenden als FHTW-Drohne bezeichnet), die mit Hilfe einer Smartphone-Applikation steuerbar ist. Diese Arbeit beschäftigt sich speziell mit der Programmierung einzelner Komponenten, die beim Bau dieser Drohne genutzt wurden. Für Details im Bereich der Mechanik und Elektronik, die auch Teil dieses Projektes waren sei auf die Arbeit meines Kollegen, Herrn Otrebski, verwiesen.

2 Stand der Technik

Im Folgenden wird ein kurzer Überblick über den derzeitigen Stand der Technik im Bereich Multicopter und Mulitcoptersteuerung gegeben.

2.1 Multicopter

Drohnen in Form von Multicoptern sind heutzutage aus der Welt des Modellbaues nicht mehr weg zu denken. Seit wenigen Jahren boomt der Einsatz von Flugdrohnen im professionellen Filmbereich. Firmen wie campilots GmbH (2014) und DIGICOPTER® Luftaufnahmen & Videoproduktion (2014) zeigen wie vielfältig der Anwendungsbereich dieser stabilen Fluggeräte sein kann und wie alte Methoden für Luftfilmaufnahmen durch moderne Technik flexibler und kostengünstiger gestaltet werden können und diese nach und nach ersetzen. Ein weiteres Beispiel für den kreativen Einsatz von Multicoptertechnologie ist das Projekt Spaxels (Ars Electronica Linz GmbH, 2014). In diesem wird ein mit LEDs ausgerüsteter Quadcopterschwarm genutzt, um dynamische, dreidimensionale Figuren im Nachthimmel zu aufzubauen. Auch in der Spielzeugindustrie haben vor allem Quadcopter, auf Grund ihrer robusten Flugeigenschaften Einzug gehalten. Mittlerweile sind Multicopter nicht nur noch Modellbauern mit langjähriger Erfahrung in diesem Bereich vorenthalten, sondern auch in allen Größen und Formen auch im Spielwarenhandel erhältlich. Im Zusammenhang mit dem hier durchgeführten Projekt sind vor allem die kleineren Bauformen der Spielwarenindustrie von Interesse und sollen im Weiteren etwas genauer unter die Lupe genommen werden. Ein bezeichnendes Beispiel derartiger Geräte soll hier vorgestellt werden.

2.1.1 Hubsan X4 Drohne

Die chinesische Firma Hubsan bietet eine Quadcopter Drohne an, die mit Hilfe einer Fernsteuerung bedient wird (Shenzhen Hubsan Technology Company Limited, 2015). Hierbei handelt es sich um eine ca. Handteller großen Drohne die für einen Preis von ca. 40 EUR erhältlich ist (Amazon Europe Core S.à r.l., 2015).

Weitere technische Details zu dieser Drohne, die im Internet recherchiert wurden können in Tabelle 1 eingesehen werden (Shenzhen Hubsan Technology Company Limited, 2015; Amazon Europe Core S.à r.l., 2015).

Abbildung 1: Hubsan X4 Drohne (Quelle: modifiziert übernommen aus Shenzhen Hubsan Technology Company Limited (2015))

Gewicht	39 g
Stromversorgung	240 mAh LiPo-Akku
Verbrauchsspannung	3,7 V
Ladezeit	30 min
Flugzeit	9 min
Mittlere Reichweite	100 m
Antrieb	4 DC-Motoren ohne Eisenkern. Diese sind eine spezielle Form des Permanentmagnet DC-Motors, welche in Richtung schneller Beschleunigung und hoher Drehzahlen optimiert ist.
Regelung/Stabilisierung	6-Achs Flight-Control System mit verstellbarer Gyroskopsensitivität

Tabelle 1: Technische Details zur Hubsan X4 Drohne

2.2 Multicoptersteuerung

Innerhalb dieses Projektes eine eigene Regelung zur Stabilisierung der zu erstellenden Drohne zu entwerfen und programmieren würde den Workload dieses Projektes weit überschreiten. Es gibt bereits zahlreiche Platinen, die eine entsprechende Fluggerätstabilisierung bieten, bzw. open source Softwareprojekte, in die bereits unzählige Stunden von Know-How geflossen sind um eine derartige Stabilisierung dem Endnutzer anzubieten. Drei derartige Produkte sollen hier kurz vorgestellt werden.

2.2.1 Hobbyking - KK2.0 Multi-rotor LCD Flight Control Board

Das Flugsteuerungsbord KK2.0 für Multirotor Fluggeräte ist ein weitverbreitetes Regelungsbord zur Flugstabilisierung. Mit Hilfe eines LCD Bildschirmes können alle notwendigen Einstellungen am Bord selbst vorgenommen werden und macht dessen Nutzung auch für Einsteiger angenehm. Somit können alle Regelparameter und Fluggerätkonfigurationen bequem auch ohne einen Computer gemacht werden. Auf Grund des Bildschirmes ergibt sich für dieses Bord ein etwas höheres Gewicht, verglichen mit Konkurrenzprodukten von $21g$ (hobbyking.com®, 2015*a*).

Mit Hilfe des Bordes können bis zu acht Motoren angesteuert werden und ermöglicht durch die vorinstallierte Firmware das Steuern von Dualcopter, Tricoptern über Quadcopter bis hin zu Octocoptern in allen erdenklichen Konfigurationen. Das Bord ist dafür ausgelegt bürstenlose Motoren anzusteuern.

Bei dem verbauten Microchip handelt es sich um einen ATmega324-PA und ermöglicht das updaten der Firmware über das USBasp AVR Programming interface. Es ist außerdem mit einem dreiachsigen Gyroskopchip und einem dreiachsigen Beschleunigungssensorchip ausgestattet, was der Bordsensorik sechs Freiheitsgrade gibt.

2.2.2 Die MultiWii Software

Bei der MultiWii Software handelt es sich um ein Open-Source Projekt zur Steuerung von Funkgesteuerten Fluggeräten. Ursprünglich war die Software auf die in der Nitendo Wii Konsolensteuerung verbauten Gyroskope und Beschleunigungssensoren ausgelegt und wurde auf der Arduino Plattform entwickelt (Dubus, 2015). Mittlerweile werden jedoch schon eine Vielzahl weiterer Gyroskop- und Beschleunigungschips unterstützt was zur Entwicklung von Flight Controller Bords unterschiedlichster Hersteller geführt hat, die auf die MultiWii Software zurückgreifen.

Flyduino - NanoWii ATmega32u4 Based MultiWii FC

Die Firma Flyduino wurde 2011 gegründet und bietet elektronische Komponenten zur Multicoptersteuerung herstellt. Die von Flyduino angebotenen Flight Controller entwickelten sich in

enger Zusammenarbeit mit der MultiWii Community und viele Produkte greifen auch heute noch auf diese Software zur Flugstabilisierung zurück (Bake, 2015).

Das Produkt NanoWii ist ein Controler Board auf Basis des ATmega32u4 Prozessors und verfügt über den MPU-6050 Chip, der ein 3-Achsen Gyroskop und einen 3-Achsen Beschleunigungssensor kombiniert. Auch dieses Bord ermöglicht das Ansteuern von acht Motoren und die Steuerung von Dual- bis zu Octocoptern in vielen unterschiedlichen Konfigurationen. Im Falle dieses Controllers kann auf den Source-Code zugegriffen und wenn notwendig Veränderungen vorgenommen werden, was den Einsatz von bürstenlosen Motoren ermöglicht. Über einen seriellen Port ist zusätzlich auch ein Bluetoothmodul zur Kommunikation mit dem Board einsetzbar und über eine I2C Schnittstelle kann weitere unterstützende Sensorik angeschlossen werden.

Des Weiteren hantelt es sich hierbei um einen sehr kleinen, leichten Controller, der ohne Pins nur $4g$ auf die Waage bringt.

HobbyKing - MultiWii 328P Flight Controller w/FTDI & DSM2 Port

Einen vergleichbaren Flight Controller wird vom Hersteller HobbyKing, auf Basis des ATmega328P Microcontrollers, angeboten. Er ist mit Beschleunigungssensoren einem 3-Achsen Gyroskop einem Magnetometer und einem Barometer ausgestattet. Das Gewicht des Bordes beträgt $13.9g$ (hobbyking.com®, 2015*b*).

Auch an dieses Bord kann ein Bluetoothmodul zur Kommunikation und zusätzliche Sensorik über eine I2C Schnittstelle angeschlossen werden. Auch dieses Bord nutzt die MultiWii Software und kann ebenfalls bis zu acht Motoren ansteuern.

3 Die FHTW-Drohne

Im Folgenden soll kurz der Aufbau Drohne dargestellt werden, welche im Zuge dieses Projektes erstellten. Zuerst soll hier die Systemarchitektur vorgestellt und anschließend die ausgewählten Teilkomponenten kurz beschrieben werden. Zuletzt werden noch kurz die verschiedenen Methoden zur Datenübertragung, die innerhalb dieses Projektes verwendet wurden beschrieben.

3.1 Systemarchitektur

Die Systemarchitektur der FHTW-Drohne kann in Abbildung 2 eingesehen werden. Es wurde hierbei großen Wert auf die Modularität der einzelnen Komponenten gelegt. Dies soll einen einfachere Erweiterung des Systems für den Einsatz in der Schwarmrobotik ermöglichen.

Die Eingabe der Steuerbefehle erfolgt vom Nutzer über eine Smartphone-Applikation, die die Befehle über Bluetooth zur Drohne überträgt. Ein Bluetooth-Shield übersetzt die gesen-

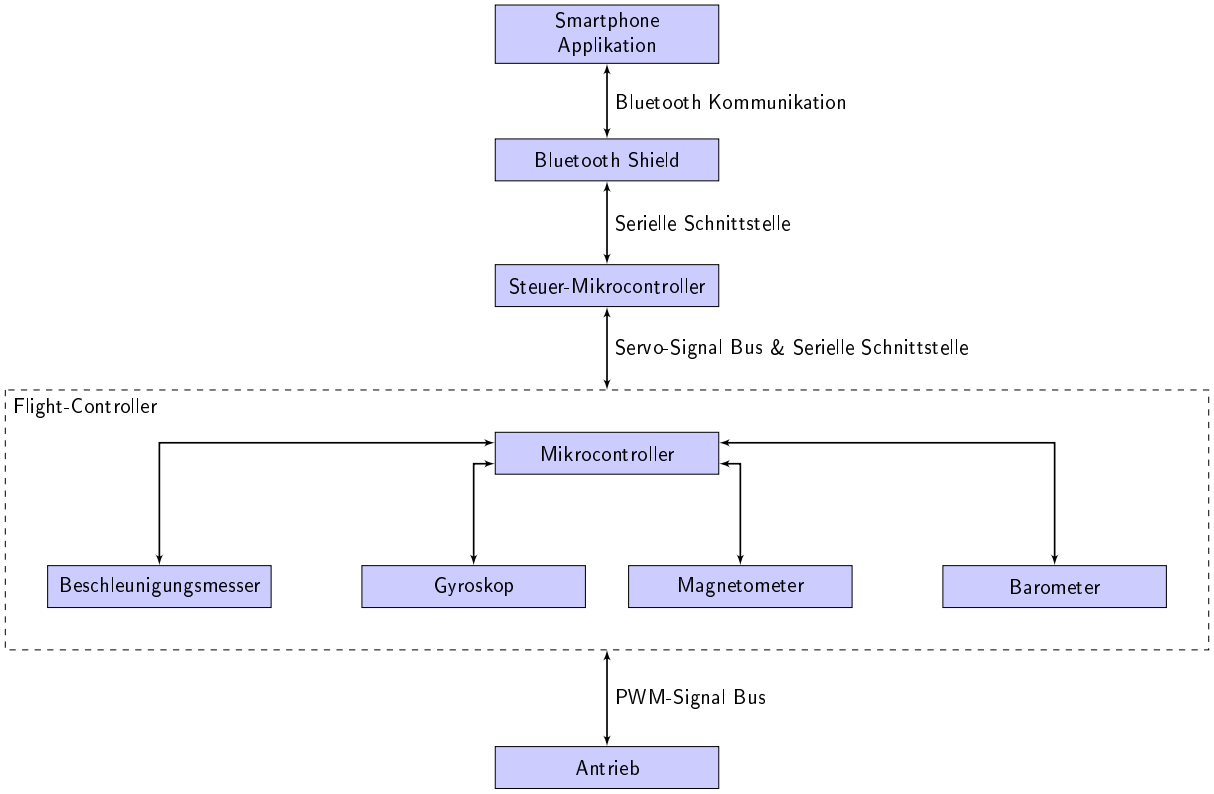

Abbildung 2: Systemarchitektur der FHTW-Drohne

deten Bluetoothsignale in serielle Datenübertragung und übermittelt diese so dem Steuer-Mikrocontroller. Dieser liest schließlich die vom Nutzer eingegebenen Steuerbefehle aus und generiert dazu passende Servosignale, die dem Flight-Controller als Steuereingaben dienen. Gesteuert können die Drehzahl der Rotoren werden (*Throttle*) und die Orientierung der Drohne. Hierbei wird ein in der Luftfahrttechnik gebräuchliches Koordinatensystem benutzt. Die Roll-Achse zeigt dabei in Flugrichtung, die Pitch-Achse zur Seite und die Yaw-Achse senkrecht nach oben, vom Fluggerät aus gesehen. Durch das Einstellen der Drehungen um diese Achsen (*Roll*, *Pitch*, *Yaw*) kann Flugrichtung der Drohne beeinflusst werden.

Zentrum der Drohnenarchitektur ist ein Flight-Controller (FC), der mit Hilfe von diverser Sensorik (*Beschleunigungssensor*, *Gyroskop*, *Magnetometer* und *Barometer*) für die Stabilisierung der Drohne im Flug sorgt und die Steuerbefehle in die entsprechende Drohnenbewegung umsetzt. Dazu steuert er die Motoren über einen PWM-Signal Bus an und gibt damit deren Drehzahl vor.

Zusätzlich ist eine serielle Kommunikation vom Steuer-Mikrocontroller mit dem FC und dessen Sensorik möglich. Damit können alle Sensordaten, die der FC verarbeitet auch für etwaige, in der Zukunft erstellte, Drohnenintelligenzen genutzt werden. Des Weiteren kann weitere Peripherie mit Hilfe des I^2C Kommunikationsprotokolls an den Steuer-Mikrocontroller angeschlossen werden. Dieser modulare Aufbau soll eine einfache Umsetzung von zukünftigem, autonomen Flugverhaltens garantieren und die komplizierten Steuerungsroutinen im FC unangetastet

lassen.

3.2 Ausgewählte Komponenten

In Abbildung 2 wurde die Systemzusammensetzung der FHTW-Drohne gezeigt. Im Folgenden werden die ausgewählten Systemkomponenten kurz vorgestellt.

3.2.1 Flight-Controller

Der FC ist das Herzstück des Unmanned Aerial Vehicle (UAV). Er kombiniert einen Mikrocontroller und mehrere Sensoren zu einer funktionalen Einheit. Der in diesem Projekt verwendete FC ist der bereits in Kapitel 2 kurz vorgestellte *MultiWii 328P Flight Controller* von Hobbyking. Folgende sind die Hauptkomponenten des Boards:

- Atmega328P Mikrocontroller
- ITG3205 dreiachsiges Gyroskop
- BMA180 Beshleunigungssensor
- HMC5883L Magnetometer
- BMP085 Barometer

Beim Layouten des FC sind dem Hersteller jedoch mehrere kleine Fehler unterlaufen, die dazu führen, dass einige Pins falsch – also nicht wie am Board angegeben – oder gar nicht verbunden sind. Näheres dazu kann in der Arbeit meines Kollegen nachgelesen werden.

Das verwendete Steuerprogramm auf diesem Board ist die MultiWii Software in der Version 2.1. Diese kann heruntergeladen und nach entsprechender Anpassung auf den Mikrocontroller gespielt werden. Alle notwendigen Einstellungen wie beispielsweise die PID Regelparameter oder Funktion von Auxiliary Pins können mit der MultiWii Kontrollsoftware auf einem PC gemacht werden.

Die Steuerbefehle können entweder über die entsprechenden Pins mit Hilfe von Servostellsignalen, oder über die serielle Kommunikationsschnittstelle direkt übergeben werden. In diesem Projekt wurden Servosignale zur Eingabe der Steuerbefehle in den FC gewählt. Schließlich werden von der Software Motorsteuersignale in Form von Pulsweitenmodulation (PWM) ausgegeben. Über eine geeignete Motortreiberschaltung kann so die Drehzahl der Motoren der Drohne gesteuert werden.

Optional kann das Board noch um einem GPS-Sender, optische Sensoren oder ein Sonar erweitert werden.

3.2.2 Steuer-Mikrocontroller

Als Steuer-Mikrocontroller wurde das Arduino Micro Board gewählt. Auf diesem ist ein ATmega32u4 Mikrocontroller mit 20 digitalen Input- und Outputpins verbaut. 7 Dieser Pins können als PWM Ausgänge verwendet werden (Pins 3, 5, 6, 9, 10, 11, 13; siehe Abbildung 3). Serielle Kommunikation kann über die Pins 1/TX und 0/RX (vgl. Abbildung 3) hardwaremäßig stattfinden. Des Weiteren verfügt das Board über einen 16 MHz Oszillator und einen Micro-USB Anschluss, über den der Mikrocontroller bequem programmiert werden kann (Arduino, 2015*a*).

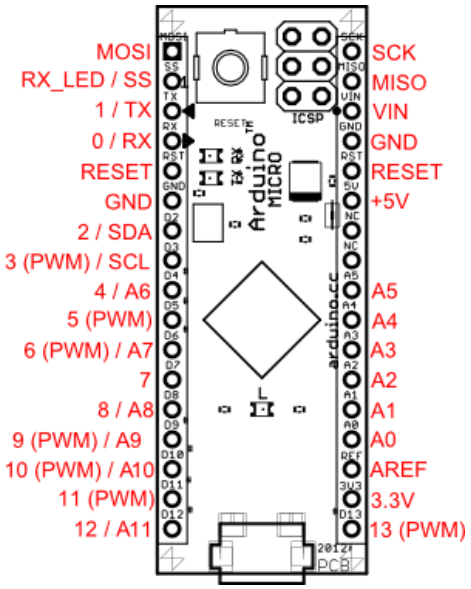

Abbildung 3: Arduino Micro (Quelle: Arduino (2015*a*))

Innerhalb dieses Projektes wird das Arduino Micro Board zur Übersetzung der vom Nutzer gesendeten Steuersignale in einen entsprechenden Servosigaloutput genutzt. Weiters ist durch den Einsatz dieses Mikrocontrollers das einfache Hinzufügen einerseits von Drohnenintelligenz möglich. So könnten beispielsweise, abhängig von den Sensordaten, Steuereingaben des Users verworfen und überschrieben werden um eine Kollision zu verhindern oder das Fluggerät bei geringer Akkuleistung zum Landen zu zwingen. Auch diverse Konzepte für Schwarmintelligenzen können hierauf programmiert werden.

3.2.3 Das Bluetooth-Shield

Um eine Kommunikation zwischen der Smartphone-Applikation und dem Steuer-Mikrocontroller zu ermöglichen wurde das Bluetoothmodul HC-06 verwendet. Mit seinem geringen Stromverbrauch ist es ein ideales Kommunikationsmodul für die FHTW-Drohne.

Voreingestellt ist die Geräte-ID *HC-06*, eine Baudrate von *9600 Bits pro Sekunde (bps)* und das Pairing-Passwort *'1234'*. Diese Einstellungen können jedoch, wenn erwünscht, jederzeit geändert werden. Dazu müssen entsprechende Befehlssätze über die serielle Schnittstelle ge-

sendet werden. Nähre Informationen können hierzu dem Datenblatt (Guangzhou HC Information Technology Co., Ltd., 2014) entnommen werden. Baudraten von *1200 bps* bis zu *115200 bps* werden von diesem Bluetoothmodul unterstützt. Im Zuge dieses Projektes wurden die Standardeinstellungen jedoch nicht verändert.

3.3 Datenübertragung

Dieser Abschnitt behandelt die verschiedenen Methoden zur Datenübertragung mit denen im Zuge der Erstellung der FHTW-Drohne gearbeitet wurde. Die einzelnen Methoden werden hier kurz eingeführt, einerseits der Vollständigkeit halber, andererseits um später auftretende Problematiken, die im Zusammenhang mit der Programmierung entstanden sind, verständlich zu machen.

3.3.1 Pulsweitenmodulation und Servosignal

Bei der PWM handelt es sich um ein Rechtecksignal, das sich innerhalb einer bestimmten Periodendauer T wiederholt (siehe Abbildung 4). Dabei gibt der so genannte Duty-Cycle die Breite des Rechtecksignals an. Der Duty-Cycle bestimmt daher das Verhältnis von T_{on} zu T. In Abbildung 4 ist ein PWM-Signal mit einem Duty-Cycle von $12{,}5\,\%$ zu sehen.

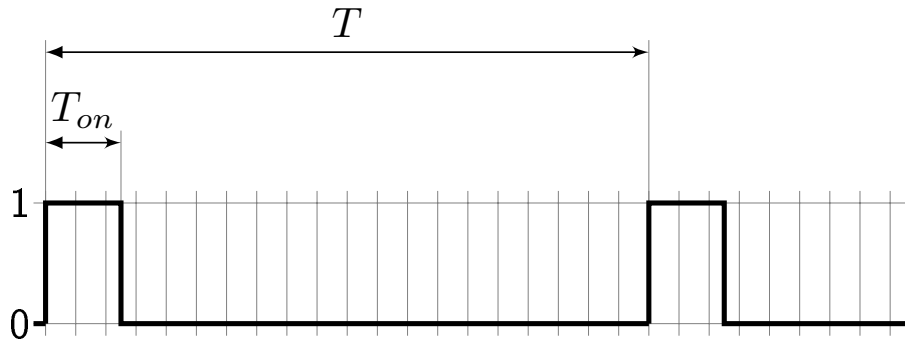

Abbildung 4: PWM-Signal mit einem Duty-Cycle von $12{,}5\,\%$.

Über PWM können so Motoren geregelt, aber auch Servos gestellt werden. Zum Stellen von Servos wird ein ganz spezielles PWM-Signal verwendet. Dabei wird ein Signal mit einer Periodendauer T von $20\ ms$ erwartet. Abhängig von der Impulsbreite zu Beginn der Periode T_{on}, können verschiedene Positionen eines Servos angesteuert werden. Eine Impulsbreite zwischen $1000\ \mu s$ und $2000\ \mu s$ gibt die Winkelstellung eines Servos zwischen $-90°$ und $+90°$ vor. Bei bei einer Impulsbreite von $1500\ \mu s$ nimmt ein angesteuerte Servo seine Mittelstellung ein.

Servosignale werden im Zuge dieses Projektes verwendet um den Steuerinput an den FC weiterzugeben.

3.3.2 Serielle Kommunikation - UART

UART ist die Abkürzung für **U**niversal **A**synchronous **R**eceiver **T**ransmitter. Es handelt sich hierbei um eine zweidrahtige Kommunikationsschnittstelle, wobei eine Leitung zum Senden und eine Leitung zum Empfangen von Daten genutzt wird. Bei der asynchronen Kommunikation über diese Schnittstelle wird, anders als bei anderen seriellen Verfahren, die Datenübertragung nicht getaktet. Bei Universal Asynchronous Receiver Transmitter (UART) werden Informationen in einem ganz speziellen Datenrahmen, bei vorgegebenen Übertragungsraten übertragen.

Die Übertragung startet daher immer mit einem *Startbit*, mit dem der weitere Datenfluss synchronisiert wird. Gefolgt wird das Startbit dann von 7 bis 8 *Datenbits*, mit denen die notwendigen Daten übertragen werden. Zur Erkennung von Übertragungsfehlern kann schließlich noch ein *Parity-Bit* gesendet werden. Die Kommunikation wird anschließend mit dem Senden eines *Stopbits* beendet. Um einen einwandfreien Datenaustausch zwischen Sender und Empfänger zu garantieren muss mit derselben Baudrate bei beiden Teilnehmern gearbeitet werden. Des Weiteren müssen die beiden Datenleitungen zwischen den beiden kommunizierenden Geräten ausgekreuzt werden: Tx (Transmitter Leitung) des Senders auf Rx (Receiver Leitung) des Empfängers und umgekehrt.

3.3.3 Bluetooth

Bluetooth ist ein in den 1990er-Jahren entwickelter Industriestandard gemäß IEEE 102.15.1, der für die Datenübertragung zwischen Geräten über kurze Distanzen verwendet wird.

Gesendet wird in einem lizenzfreien ISM-Band (Industrial, Scientific, Medical Band) zwischen $2,402GHz$ und $2,480GHz$. Dieses Frequenzband ist weltweit frei verwendbar, kann aber unter anderem von WLANs oder Mikrowellenherden, die im selben Band arbeiten, gestört werden. Die Reichweite der Bluetooth Datenübertragung ist einerseits von der Sendeleistung abhängig (bis zu $100m$ Reichweite möglich), andererseits aber von vielen anderen Faktoren, wie zum Beispiel Hindernisse innerhalb der Funkstrecke.

4 Programmierung

In diesem Kapitel soll näher auf die unterschiedlichen Funktionsteile des Projekts und deren Programmierung eingegangen werden. Hierbei wird strukturiert vorgegangen und mit dem FC begonnen werden, auf den dann der Arduino Mini aufbaut. Dieser wiederum kommuniziert über Bluetooth mit einer Mobiltelefonapplikation. Die Programmierung dieser Komponenten soll in entsprechender Reihenfolge beschrieben werden. Dazu werden Codeausschnitte gezeigt, die vom Autor für das Verständnis der Programmierung für besonders wichtig gehalten werden.

4.1 Der Flight-Controller

Da der FC die MultiWii Steuerungssoftware zur Regelung der Motoren nutzt, ist der Programmieraufwand hier relativ gering. Es müssen lediglich ein paar Setupeinstellungen in der Konfigurationsdatei (`config.h`) der MultiWii Software gemacht werden.

Zuerst sollte der Multicopter-Typ eingestellt werden. Abhängig von der Anzahl der Motoren und der Geometrie des Fluggerätes sind verschiedene Konfigurationen möglich. Nähere Informationen zu den verschiedenen Ausprägungen von Multicoptern können in der Arbeite meines Kollegen nachgelesen werden. Im Falle der FHTW-Drohne, die mir vier Motoren arbeitet sind zwei Konfigurationen realisierbar: *Quad+* und *QuadX*. Bei *Quad+* liegt die Vorwärtsrichtung fluchtend mit einem Ausleger für die Motoren. Bei *QuadX* liegt die Vorwärtsrichtung zwischen zwei Auslegern. In letzterer Konfiguration wurde für die FHTW-Drohne umgesetzt. In der Konfigurationsdatei kann die durch ein `define` bequem eingestellt werden (siehe Quellcode 1, Zeile 3). Des Weiteren können die Minimale und Maximale Motorgeschwindigkeit in der Konfigurationsdatei eingestellt werden. Dies wird durch Festlegung des kleinsten und größten Servosignalimpulses erreicht. Der Wertebereich der hier zur Verfügung steht sind $1000\ \mu s$ bis $2000\ \mu s$, analog zur Steuerung von Servos. Der Minimalwert ist die kleinste Motorgeschwindigkeit, die der FC beim Flug einstellen wird. Daher sollten sich die Rotoren bei dieser Geschwindigkeit noch bewegen. Diese werden in der Datei `conf.h` durch `defines` fixiert (siehe Quellcode 1, Zeilen 6 und 7). Auch die Baudrate, die für die Kommunikation mit dem FC genutzt wird kann hier adaptiert werden. Wichtig ist hierbei, das zu kleine Baudraten zu Verzögerungen in der Motorregelung während der Kommunikation führen können. Dies sollte tunlichst vermieden werden, weshalb in diesem Projekt eine Baudrate von *115200 bps* gewählt wurde. Das für diese Einstellung notwendige `define` ist im Quellcode 1 in Zeile 10 zu sehen. Zu guter Letzt soll hier noch die Einstellung der Orientierung des Magnetometers in Codezeile 13 gezeigt werden. Dies ist dann notwendig, wenn die Orientierung des Sensors nicht der des FCs entspricht.

```
1  /*** The type of multicopter ***/
2  //#define QUADP
3  #define QUADX
4
5  /*** Motor minthrottle/maxthrottle ***/
6  #define MINTHROTTLE 1150
7  #define MAXTHROTTLE 1850
8
9  /*** Serial com speed ***/
10 #define SERIAL_COM_SPEED 115200
11
12 /*** individual sensor orientation ***/
13 #define MAG_ORIENTATION(X, Y, Z) {magADC[ROLL] = -Y; magADC[PITCH] = X;
       magADC[YAW] = Z;}
```

Quellcode 1: Ausschnitte aus der MultiWii Konfigurationsdatei conf.h

In der Konfigurationsdatei `conf.h` können natürlich noch eine Vielzahl weiterer Setupeinstellungen getroffen werden, wie zum Beispiel die Kommunikationsgeschwindigkeit des I^2C Busses oder die Orientierung des Magnetometers (falls der FC in einer anderen als der Standardorientierung am Fluggerät angebracht wird). Im Zuge dieses Projektes wurden noch `#define I2C_SPEED 400000L`, `#define INTERNAL_I2C_PULLUPS`, `#define RCAUXPIN12` und `#define FREEIMUv035_BMP` gesetzt, alle anderen Einstellungen bei ihren Standardwerten belassen.

4.2 Arduino Micro

Wie bereits beschrieben kommuniziert der Adruino Micro mit dem Nutzer über Bluetooth. Dazu steht ein Bluetoothmodul zur Verfügung, das über eine serielle Schnittstelle entsprechende Informationen an den Mikrocontroller weiter leitet. Diese müssen anschließend vom Controller in entsprechende Steuerbefehle an den FC umgewandelt werden. Ein entscheidender Punkt hierbei ist, dass alle Befehle in einem kontinuierlichen Fluss ankommen. Entsprechend muss auch das Programm ausgelegt sein.

Zusätzlich soll auch eine serielle Kommunikation mit dem FC ermöglicht werden. Leider bietet das Arduino Micro Board jedoch nur eine einzige hardwareseitige UART Kommunikationsschnittstelle welche bereits für den Informationsaustausch mit dem Bluetooth-Shield verwendet wird. Es muss daher auf eine softwaremäßige Umsetzung der seriellen Kommunikation zurückgegriffen werden.

4.2.1 Steuerbefehl-Protokoll

Um mit einen kontinuierlichen Befehlsfluss entsprechend umgehen zu können wurde ein Protokoll erstellt, mit dem Befehlssätze versandt und in entsprechende Steuerbefehle umgewandelt werden. Dabei ist der Befehlsstring wie folgt definiert: `*[Code]|[Wert]#` und kann in folgende Zeichen zerlegt werden:

- `*` - Markierung für den Beginn des Befehlsstrings
- `[Code]` - Die Codenummer, die den auszuführenden Befehl bestimmt
- `|` - Trennzeichen zwischen Codenummer und Befehlswert
- `[Wert]` - Der Wert, der dem entsprechenden Befehl übermittelt wird
- `#` - Markierung für das Ende des Befehlsstrings

Die Codenummer, deren Wertebereich und die Beschreibung derselben kann in Tabelle 2 eingesehen werden.

Unter Verwendung dieses Protokolls können die Steuerbefehle nacheinander ausgelesen und entsprechende Einstellungen an den Ausgabepins gemacht werden. Die Decodierung des

Code	Befehl	Datenbereich	Beschreibung
1	THROTTLE	$0 - 1{,}000$	
2	ROLL	$-500 - 500$	
3	PITCH	$-500 - 500$	
4	YAW	$-500 - 500$	
5	ARM	–	Der FC gibt die Motoren für den Start frei
6	DISARM	–	Der FC sperrt die Motoren
7	HOLD ON	–	Der FC sorgt dafür das die Drohne ihre Höhe hält
8	HOLD OFF	–	Das automatische halten der Höhe wird wieder deaktiviert

Tabelle 2: Auflistung der implementierten Befehle.

vom Bluetooth-Shield übermittelten Befehlsstrings wird in Qellcode 2 gezeigt. In der Hauptschleife des Programmes am Steuer-Miktrcontroller wird kontinuierlich überprüft ob neue Daten ankommen (Codezeile 4). Ist dies der Fall wird das empfangene Byte ausgelesen (Codezeile 5) und überprüft ob es sich hierbei um den vereinbarten Charakter handelt, der den Beginn des Befehlssatzes markiert (Codezeile 7). Wenn ja, werden die Codenummer für den auszuführenden Befehl und der mit diesem Befehl verknüpfte Wert ausgelesen (Codezeilen 8, 9). Abhängig von der Befehlsnummer werden schließlich die Ausgangspins des Mikrocontrollers gesetzt (Codezeile 11). Wird die Bedingung einer der beiden beiden hier beschrieben Abfragen (Codezeile 4, 7) nicht erfüllt wird die Hauptprogrammschleife erneut gestartet.

```
1 #define START_CMD_CHAR '*'
2
3 void loop() {
4    if(Serial1.abailable() < 1) return;
5    get_char = (char)Serial1.read();
6
7    if (get_char != START_CMD_CHAR) return;
8    command = Serial1.parseInt();
9    value = Serial1.parseInt();
10
11   switch(command) {
12   ...
13   }
14 }
```

Quellcode 2: Dekodierung der Steuerbefehle am Arudino Micro

An den Ausgangspins, die einen Steueroutput – *Throttle*, *Roll*, *Pitch*, *Yaw* und *Auxiliary Pins* zum Armen/Disarmen des Fluggerätes und starten des automatischen Höhe haltens (vgl. Tabelle 2 Steuerbefehle) – benötigen, werden anschließend Servosignale generiert. Dies gelingt sehr einfach mit der Arduino Bibliothek *Servo* (Arduino, 2015*b*). Auf ein Codebeispiel zu dieser Standardbibliothek von Arduino wird innerhalb dieser Ausarbeitung verzichtet. Es soll hier auf den kommentierten Programmcode verweisen werden, der zusammen mit dieser Arbeit abgegeben wurde, zu finden ist.

4.2.2 Softwaremäßige Serielle Kommunikation

Üblicherweise findet UART Kommunikation mit über ein eigenständiges Bauelement (einem UART-Chip bzw. -Baustein) oder einem eigenen Funktionsblock auf einem Mikrocontroller statt. Das verwendete Arduino Micro Board bietet aber, auf Grund seiner kleinen Bauweise, nur eine einzige derartige Kommunkiationsmöglichkeit: Pins 1/TX und 0/RX (siehe Abbildung 3).

Glücklicherweise bietet die Arduino Bibliothek *SoftwareSerial* eine softwaretechnische Implementierung der UART Kommunikation (Arduino, 2015*c*). Damit ist eine serielle Kommunikation mit Übertragungsgeschwindigkeiten bis zu *115200 bps* auch mit über andere Pins möglich. Die Initialisierung einer Kommunikationsverbindung mit Hilfe der *SoftwareSerial* Bibliothek wird in Quellcode 3 gezeigt. Durch Anschluss der Datenleitungen an die im Code angegebene Pins ist so ein Datenaustausch möglich.

```
1 #include <SoftwareSerial.h>
2 SoftwareSerial SerialS(8,9); // RX, TX
3 void setup() {
4    SerialS.begin(115200);
5 }
```

Quellcode 3: Initialisierung der SoftwareSerial Verbindung

Im Zuge er Programmierung wurde jedoch bei der Nutzung der *SoftwareSerial* Bibliothek eine Beeinflussung der Servosignale, die die Steuerbefehle an den FC weitergeben, festgestellt. Sowohl die verwendete *Servo*, als auch die *SoftwareSerial* Bibliothek verwenden Interrupts, die sich gegenseitig überschneiden und somit ein rechtzeitiges Auslösen behindern. Dadurch werden die generierten Servosignale beeinflusst, was zu einem zittern derselben (also unregelmäßige Impulsbreiten; vgl. Abschnitt 3.3.1) und einem Verfälschen der vom User eingegebenen Steuersignale führt.

Abhilfe schafft die alternative Bibliothek *AltSoftSerial*, entwickelt von Paul Stoffregen (PJRC.COM, 2015). Unter Verwendung dieser Umsetzung einer softwaremäßigen UART Kommunikation treten genannte Beeinflussungen nicht mehr auf. Die Pins die zur Signalübertragung genutzt werden können sind jedoch fix und können nicht selbst gewählt werden. Im Falle des verwendeten Arduino Micro Boardes steht bei Verwendung dieser Bibliothek nur eine softwareseitige serielle Verbindung zur Verfügung, wobei die Übertragungsleitung TX an Pin 5 und

die Empfangsleitung auf Pin 13 angeschlossen werden müssen. Die Initialisierung erfolgt sehr ähnlich zum Standard SoftwareSerial (siehe Quellcode 4).

```
1 #include <AltSoftSerial.h>
2 SoftwareSerial SerialS; // RX 5, TX 13
3 void setup() {
4    SerialS.begin(115200);
5 }
```

Quellcode 4: Initialisierung der alternativen SoftwareSerial Verbindung

Damit ist also eine serielle Kommunikation sowohl mit dem Bluetooth-Shield, als auch mit dem FC möglich.

4.2.3 Abfrage Sensordaten

Durch das Nutzen einer softwaremäßigen UART-Implementierung ist eine serielle Kommunikation mit dem FC und ein einfaches Abfragen der Sensordaten der Sensoren am Controllerboard möglich. Diese können dann vom Steuer-Mikrocontroller für die interne Intelligenz der Drohne weiterverwendet werden. Hierbei sind zwei Arten von Nachrichten zu unterscheiden:

- Anfragen, die zum FC gesendet werden und
- Ausgabenachrichten vom FC-Board.

Die Anfrage-Nachrichten, mit denen nach dem MultiWii Serial Protocol (MSP) Daten angefragt, spezielle Befehle gesendet und neue Parameter in die MultiWii Steuerung eingegeben werden können sind und die Antwort-Nachrichten vom FC sind folgendermaßen aufgebaut:

- Anfrage: `$M<[data length][code][data][checksum]`
- Antwort auf bekannte Anfrage: `$M>[data length][code][data][checksum]`
- Antwort auf unbekannte Anfrage: `$M|[0][code][checksum]`

Alle Teile der Nachricht werden, abgesehen vom `[data]`-Teil, als einzelne Bytes gesendet. Das `[data length]`-Byte gibt die Länge der übertragenen Daten an, das `[code]`-Byte Nachrichten-Code der angefragten Daten (siehe Tabelle 3). Der *[data]*-Teil der Nachricht kann auch aus mehreren Bytes bestehen. Alle gesendeten Nachrichten werden mit einem `[checksum]`-Byte abgeschlossen. Dadurch kann festgestellt werden, ob Fehler in der Übertragung stattgefunden haben. Die Checksumme errechnet sich hierbei aus den XOR-Verknüpfungen des `[data length]`-Bytes mit dem `[code]`-Byte und allen `[data]`-Bytes. Die Funktion zur Kommunikation nach dem MSP wird im Quellcode 5 gezeigt. Die Berechnung des `[checksum]`-Bytes ist in Zeilen 6, 9 und 13 zu sehen.

Kommando	Code	Data	Datentyp	Beschreibung
MSP_ATTITUDE	108	`angx`	INT16	Orientierungswinkel um die x-Achse der Drohne; Datenbereich [-1800, 1800] (Einheit: $\frac{1}{10}^\circ$)
		`angy`	INT16	Orientierungswinkel um die y-Achse; Datenbereich [-900, 900] (Einheit: $\frac{2}{10}^\circ$)
		`heading`	INT16	Orientierungswinkel um die z-Achse; Datenbereich [-180°, 180°]
MSP_ALTITUDE	109	`EstAlt`	INT32	geschätzte Flughöhe in cm
		`vario`	INT16	Veränderung der Flughöhe in $\frac{cm}{s}$

Tabelle 3: Beispiele für MSP Nachrichtencodierung

```
1 void send_msp(uint8_t opcode, uint8_t * data, uint8_t n_bytes) {
2    uint8_t checksum = 0;
3    // Send the MSP header and message length
4    SerialS.write((byte *)"$M<", 3);
5    SerialS.write(n_bytes);
6    checksum ^= n_bytes;
7    // Send the code byte
8    SerialS.write(opcode);
9    checksum ^= opcode;
10   // Send the data bytes
11   for(int i = 0; i < n_bytes; i++) {
12      SerialS.write(data[i]);
13      checksum ^= data[i];
14   }
15    // Send the checksum
16   SerialS.write(checksum);
17 }
```

Quellcode 5: Funktion zur Kommunikation nach dem MSP

Die Liste an Nachrichten-Codes ist lange und umfasst nicht nur das Auslesen der rohen Sensordaten, sondern auch bereits interpretierter Daten, wie die Ausrichtung des Bordes oder die geschätzte Flughöhe. Auch können Steuerbefehle gesendet und Regelparameter gesetzt werden. In Tabelle 3 werden zwei Codes des MSP gezeigt, mit denen die Orientierung der Drohne und deren Flughöhe abgefragt werden können:

Nach dem Senden einer entsprechenden Anfrage zum FC mit dem Richtigen Nachrichtencode, wird ein Antwort nach zuvor beschriebener Codierung zurückgegeben. Dabei ist der `[data]`-Teil der Nachricht üblicherweise größer als ein Byte. In den in Tabelle 3 gegebenen Beispielen hat er eine Größe von genau 6 Bytes. Die gesuchten Daten (zB die Orientierung um die z-Achse) müssen daher aus den übertragenen 6 Bytes des `[data]`-Teiles ausgelesen wer-

den. Alle übermittelten Daten sind hierbei Little-Endian. Dies bedeutet, dass das kleinstwertige Byte zuerst übertragen und das höchstwertige Byte zuletzt. Es soll ausdrücklich darauf hingewiesen werden, dass die Byte-Reihenfolge der angefragten Daten von besonderer Wichtigkeit ist. Werden die Daten falsch ausgelesen, ergeben sich unsinnige Werte.

4.3 Die Handyapplikation

Die Handyapplikation wurde für Android Mobiltelefone ausgelegt und somit auf der Android Plattform programmiert. Bevor hier auf das erstellte Programm selbst eingegangen wird, soll die grundlegende Systemarchitektur vorgestellt werden.

4.3.1 Systemarchitektur

Die Systemarchitektur ist ein Softwarestapel in dem jede Ebene aus mehreren einzelnen Programmen besteht, die verschiedene Systemfunktionen zur Verfügung stellen. Wie in Abbildung 5 zeigt besteht die Architektur aus 5 Ebenen: Den Linux Kernel und seine Low-level Tools (rot), verschiedenste Bibliotheken (grün), die Android Runtime (gelb) und schließlich das Anwendungsframework und die Applikationen selbst (blau) (Künneth, 2012).

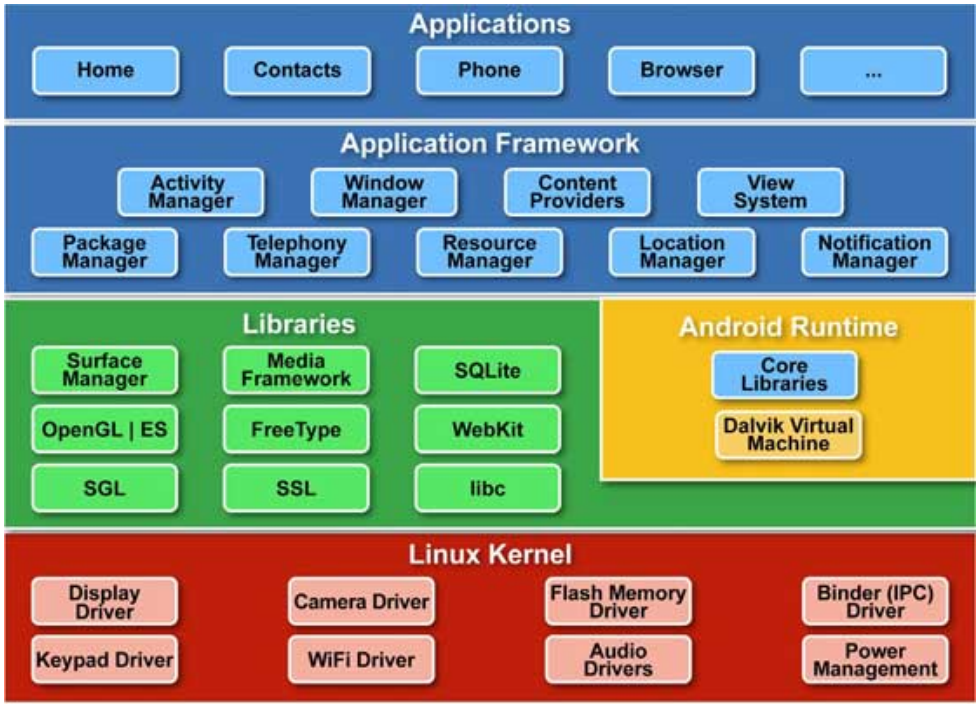

Abbildung 5: Android Systemarchitektur (Quelle: Tutorials Point (2015))

Linux Kernel

Basis der Android Architektur ist der Linux 2.6 Kernel, der als Hardwareabstraktionsebene genutzt wird. Vom Linux Kernel werden die Basissystemfunktionalitäten zur Verfügung gestellt. Darunter fallen unter anderem das Prozessmanagement, das Speichermanagement und das Gerätemanagement um das Benutzen der Kamera, der Tastatur oder des Touchscreens zu ermöglichen. Des Weiteren handhabt der Kernel auch Aufgaben wie Netzwerkkommunikation und stellt eine Vielzahl an Hardwaretreibern bereit.

Bibliotheken

Auf den Linux Kernel aufgesetzt sind unterschiedlichste Bibliotheken. Diese sind in den Programmiersprachen C und C++ geschrieben, bereits kompiliert und auf den entsprechenden Geräten bereits vorinstalliert. Applikationsentwickler greifen auf diese indirekt über das Android-Anwendungsframwork zu. Zur Verfügung stehen beispielsweise:

- *Media Libraries*, die das Aufnehmen und Wiedergeben von Audio- und Videomaterial ermöglichen,
- ein *Surface Manager* für den Bildschirmzugriff und somit eine entsprechende visuelle Ausgabe der anzuzeigenden Daten,
- *SQLite*, eine relationale Datenbank, mit der Daten geordnet abgelegt und wieder zugegriffen werden können,
- die *SystemC Library*, mit der die Standard C-Bibliothek, speziell auf mobile Geräte angepasst, etc.

Android Runtime

Der Großteil der vorzufindenden Android-Applikationen ist in der Programmiersprache Java programmiert. Um Javacode ausführen zu können ist eine so genannte Laufzeitumgebung notwendig, die es ermöglicht den Code weitgehend plattformunabhängig auszuführen. Diese Laufzeitumgebung (engl. Runtime Environment) stellt eine Softwareplattform dar und definiert Anwendungsprogrammierschnittstellen (APIs) und enthält eine virtuelle Maschine auf der das entsprechende Programm schließlich ausgeführt wird.

Die in der Android-Architektur verwendete virtuell Maschine wird *Dalvik* genannt und wurde von Dan Bornstein ausdrücklich für den Einsatz in mobilen Geräte konzipiert. Damit gibt es einige wesentliche Unterschiede zu den klassischen Java-Laufzeitumgebungen. Dalvik wurde einerseits auf einen minimalen Speicherverbrauch hin optimiert, andererseits wird jede Android-Anwendung als Prozess in einer eigenen Instanz der Dalvik Virutal Machine ausgeführt. Damit soll sowohl die Stabilität als auch die Sicherheit der Plattform gesteigert werden.

Die *Core Libraries* sind ein weiterer Teil der Android Runtime und beinhaltet wichtige Java-Pakete: `java.lang`, `java.io`, `java.math`, `java.util`, etc. Diese Pakete enthalten wichtige Klassenbibliotheken die dann im Code genutzt werden können.

Anwendungsframework

Das Anwendungsframework bietet einen großen Funktionsumfang, mit dessen Hilfe mobile Anwendungen bequem erstellt werden können. Mit ihnen bekommt der Applikations-Entwickler einfachen Zugriff auf die Gerätehardware wie beispielsweise die Kamera (sofern vorhanden), Kommunikationsmodule (zB Bluetooth) oder interne Sensoren. Aber auch das Lesen und Schreiben von Informationen ist möglich, wie zum Beispiel Kontaktdaten oder Termine. Der Umfang, in dem die Applikation auf diese Funktionen zugreifen darf wird durch ein Rechtesystem gesteuert.

Ein wichtiges Kernkonzept dieses Frameworks ist das die eigene Programmfunktion auch anderen Programmen zur Verfügung gestellt wird. So kann jede Applikation sowohl die Funktion anderer Anwendungen anfordern, aber auch die eigene anbieten.

Die Hauptbestandteile des Anwendungsframeworks sind:

- *Views* sind die Grundsteine aus denen die Benutzeroberflächen zusammengestellt sind. Neben einer Vielzahl an Standardelementen wie Textfelder, Buttons, Listen, etc., können bestehende Views auch individuell angepasst oder komplett eigenständig erstellt werden.
- Mit Hilfe des *Content Providers* kann jede Anwendung auf Daten anderer Programme zugreifen oder selbst Daten bereitstellen.
- Der *Resource Manager* ermöglicht den Zugriff auf lokale Grafiken, Layoutdatein, etc.
- Mit dem *Notification Manager* kann die Anwendung auf die Android-Statuszeile zugreifen und beispielsweise Popup-Nachrichten anzeigen.
- Der *Activiy Manager* ist für die Steuerung des Lebenszyklus der Applikation zuständig.

Android Applikationen

Hier sind selbst erstellte, als auch alle auf dem Gerät installierten Anwendungen zu finden. Sie stellen ihre eigene Funktion sowohl dem Nutzer als auch anderen Applikationen zur Verfügung.

4.3.2 Generelle Programmstruktur von Applikationen

Eine Handyapplikation setzt sich aus vielen, miteinander verknüpften Dateien zusammen. Die meisten integrierten Entwicklungsumgebung, die das Erstellen von Android Applikationen unterstützen, generieren ein funktionsfähiges Grundgerüst welches alle benötigter Programmdateien enthält automatisch. Dabei handelt es sich einerseits um XML-Dateien, mit der die Applikationsstruktur (Aussehen der Oberfläche, Zusammenspiel der einzelnen Aktivitäten, etc.)

festgelegt werden kann, andererseits um Java-Dateien, die den Programmcode enthalten und den einzelnen Oberflächen ihre Funktion geben. Auf die wichtigsten Programmdateien soll hier kurz eingegangen werden, um die allgemeine Struktur eines Androidprogrammes zu verdeutlichen.

Das *Manifest-File* (zB `AndroidManifest.xml`) stellt dem Android Betriebssystem alle essentiellen Informationen, über die Applikation zur Verfügung. In dieser Datei werden Berechtigungen für Hardwareperipherie erteilt und die generelle Programmstruktur beschrieben. Den einzelnen Aktivitäten (also einzelne Benutzeroberflächen in einer Applikation, die Benutzereingaben zulassen) des Programmes werden hier mit ihren Java-Programmdateien verknüpft.

Die *Codedateien* (zB `MainActivity.java`) werden in der Programmiersprache Java geschrieben. Für die Erstellung neuer Aktivitäten gibt es vorgefertigte Klassen, die entsprechend der Wünsche des Programmierers abgeändert werden können. Ein Auszug aus den möglichen Callback-Funktionen die einer Aktivität im Zuge ihres Lebenszyklus zur Verfügung stehen ist in Tabelle 4 zu sehen. Beim Start der Aktivität wird vom Programm die Benutzeroberfläche generiert, die mit Hilfe einer XML-Datei entworfen wurde. Anschließend kann auf Benutzereingaben über Views (Bedienelemente) reagiert werden.

Callback-Funktion	**Beschreibung**
`onCreate()`	Diese Funktion wird aufgerufen, wenn die Aktivität zum ersten mal gestartet wird.
`onStart()`	Diese Funktion startet, wenn die Aktivität für den Nutzer zum ersten mal sichtbar wird.
`onReturn()`	Diese Funktion startet, wenn die Aktivität wieder aktiv wird.
`onDestroy()`	Bevor die Aktivität geschlossen wird, wird diese Funktion aufgerufen.

Tabelle 4: Callback-Funktionen einer der Aktivitäten-Klasse

Prinzipiell werden aber alle Funktionen, die eine Verarbeitung durch die Applikation benötigen in Java-Code geschrieben. So können beispielsweise eigene Klassen für drahtlose Kommunikation oder individuelle Bedienelemente erstellt werden. Dabei kann auf bereits existierende Klassenstrukturen zurückgegriffen werden, die dann benutzerspezifisch abgeändert werden oder komplett neue entworfen werden.

Die *Benutzeroberflächen* werden den einzelnen Aktivitäten in Form von XML-Dateien (zB `activity_main.xml`) zur Verfügung gestellt. Dabei stehen dem Programmierer eine Reihe von Layoutvarianten zur Verfügung: *Linear Layout*, *Relative Layout*, *Table Layout*, *Absolute Layout*, *Frame Layout*, *List View*, *Grid View*. Mit Hilfe dieser Layouts können einzelne *Views* (also Buttons, Schieber, Textfelder, etc.; vgl. Abschnitt 4.3.1) nach belieben angeordnet und dem Nutzer zur Eingabe angeboten werden.

4.3.3 Bluetooth in Android

Android stellt eine eigene Bluetooth API zur Verfügung, die das drahtlose Kommunizieren und den Austausch von Daten mit anderen Bluetoothgeräten ermöglicht. Hierbei bietet das bereits erwähnte Applikationsframework Zugriff auf die Bluetoothfunktionalitäten des Smartphones über die Android Bluetooth APIs, die das Verbinden von einzelnen Android-Applikationen mit anderen Geräten, Punkt-zu-Punkt und Mehrpunkt-Features freigibt.

Eine Android-Applikation kann unter Verwendung der Bluetooth API folgende Aktionen durchführen:

- Nach anderen Bluetoothgeräten suchen
- Den lokalen, am Telefon befindlichen, Bluetoothadapter nach verbundenen Geräten fragen
- RFCOMM-Kanäle aufbauen
- Sich mit anderen Bluetoothgeräten verbinden
- Daten empfangen und senden
- Mehrfache Verbindungen managen

4.3.4 Erste Testapplikation

Für die ersten Tests wurde eine Applikation geschrieben, die das Suchen und Verbinden mit anderen Bluetoothgeräten erlaubt. Des Weiteren wurden Bedienelemente (Views) hinzugefügt, die, wenn ein anderes Gerät mit der Applikation verbunden ist, nach Anschnitt 4.2.1 codierte Befehle an das verbunden Device senden.

Die Testapplikation kann in Abbildung 6 gesehen werden. Im Activity Bar wird der Name der Applikation (*FHTW-Drohne*), sowie ein Menüpunkt angezeigt, der einen Verbindungsaufbau mit einem Bluetoothgerät ermöglicht. Nachdem eine Verbindung hergestellt wurde können die Schieberegler (*SeekBar*) verwendet werden um Steuerbefehle an den Arduino Micro zu senden. Außerdem wurden zwei Schalter (*Switch*) hinzugefügt mit denen der FC scharf gemacht/entschärft und die FC-Funktion 'Position halten' aktiviert/deaktiviert werden können.

Die Funktionstüchtigkeit der Applikation im Zusammenspiel mit dem Arduino Micro wurde getestet. Alle in diesem Programm ermöglichten Steuereingriffe werden vom Steuer-Mikrocontroller erfolgreich übersetzt und an den FC weitergegeben.

Programmaufbau und -ablauf

Im Folgenden soll kurz der Aufbau der Testapplikation erläutert werden. Hierbei werden generelle Eigenschaften der Android Applikationsprogrammierung gezeigt und nicht detailliert auf

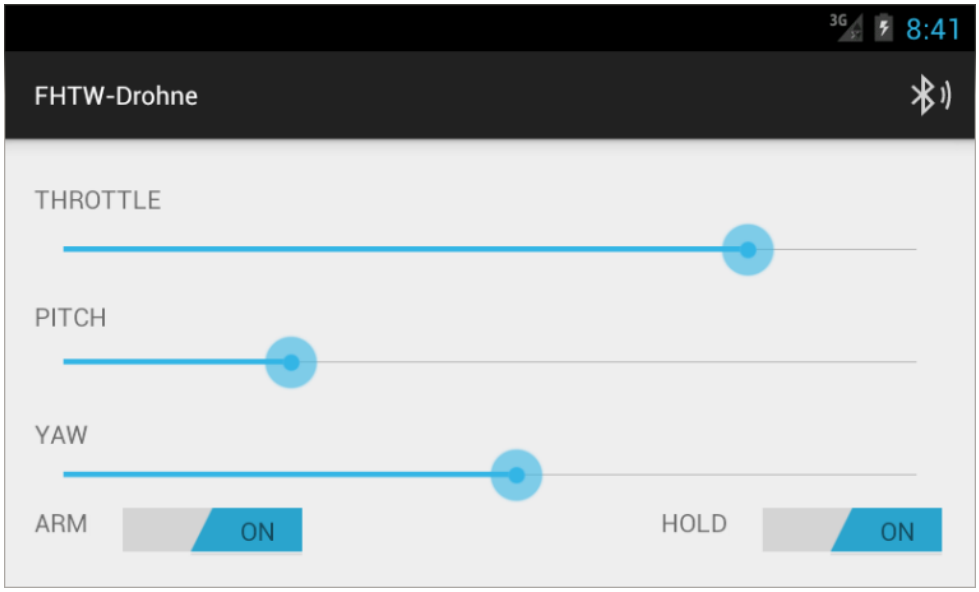

Abbildung 6: Testapplikation

den gesamten Code eingegangen. Ausschnitte des Quellcodes werden gezeigt um allgemeine Programmierkonzepte zu verdeutlichen.

Für die Bluetooth-Kommunikation wurde eine eigene Klassen-Datei `BluetoothControllService.java` geschrieben. Diese Klasse bietet Funktionen die alle für die Applikation notwendigen Bluetooth-Funktionalitäten einfach zugänglich machen: Das Suchen nach anderen Bluetooth-Geräten, das Verbinden mit anderen Geräten und das Senden und Empfangen von Daten. Quellcode 6 zeigt wie die erstellte Klasse im Hauptprogramm eingebunden wird. Dort kann, nach dem Erstellen und Initialisieren eines entsprechenden Objektes `mControlService` (Codezeilen 1, 2) auf die `write`-Funktionen des Objektes zugegriffen werden, um in der Funktion `sendCommand()` Daten über Bluetooth zu versenden.

```
1 private BluetoothControlService mControlService = null;
2 mControlService = new BluetoothControlService(getApplicationContext(), mHandler);
3
4 private void sendCommand(String command) {
5     // Get the command bytes to tell the BluetoothControlService to write
6     byte[] send = command.getBytes();
7     mControlService.write(send);
8 }
```

Quellcode 6: Einbindung des Bluetooth–Services

Beim Öffnen der Applikation wird die Hauptaktivität (`MainActivity.java`) gestartet und damit die Funktion `onCreate()` aufgerufen (siehe Quellcode 7). Zu aller erst wird die Benutzeroberfläche geladen (Codezeile 3). Anschließend wird ein Bluetooth Adapter erstellt (Zeile 5), der das Ausführen aller fundamentalen Bluetooth-Aufgaben ermöglicht und die View Objekte (Schieberegler) mit ihren Darstellungen in der Benutzeroberfläche verknüpft (Zeilen 7 – 9).

Wird letzteres versäumt kann das Java-Programm die Eingaben des Users nicht verarbeiten. Schließlich wird überprüft, ob das Smartphone, auf dem die Applikation läuft überhaupt über ein Bluetoothmodul verfügt. Ist dies nicht der Fall wird die Applikation beendet und der Nutzer über den Grund des Abbruches verständigt (Zeilen 13 – 16).

```
1 protected void onCreate(Bundle savedInstanceState) {
2    super.onCreate(savedInstanceState);
3    setContentView(R.layout.activity_main);
4    // Get local Bluetooth adapter
5    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
6
7    mSeekBar_1 = (SeekBar)findViewById(R.id.seekbar_1);
8    mSeekBar_2 = (SeekBar)findViewById(R.id.seekbar_2);
9    mSeekBar_3 = (SeekBar)findViewById(R.id.seekbar_3);
10
11   ...
12
13   if(mBluetoothAdapter == null) {
14      Toast.makeText(getApplicationContext(), "Bluetooth␣is␣not␣available",
            Toast.LENGTH_LONG).show();
15      finish();
16   }
17 }
```

Quellcode 7: Callback–Funktion onCreate() der Testapplikation

Bevor Eingaben des Benutzers überhaupt verarbeitet und an ein anderes Bluetooth-Gerät versendet werden können, muss sich der Benutzer zuerst mit einem anderen Gerät verbinden. Dies gelingt über den entsprechenden Menüpunkt am rechten Rand des Activity Bars (siehe Abbildung 6). Nach Anwählen des Bluetooth Icons kann der Nutzer nach Geräten suchen und sich mit diesen verbinden.

Nachdem eine erfolgreiche Verbindung hergestellt wurde müssen die Steuerbefehle des Benutzers versandt werden. Dies gelingt durch das Erstellen eines *Listeners*, der Veränderungen der Bedienelemente registriert und entsprechende Funktionen aufruft. Quellcode 8 zeigt einen `OnSeekBarChangeListener`, mit dem Usereingaben erkannt werden. Die aktuelle Einstellung des Schiebereglers werden ausgelesen und anschließend wird über die Funktion `sendCommand()` ein Steuerbefehl an die FHTW-Drohne versandt. Für alle weiteren Views, die eine Benutzereingabe ermöglichen muss ein Listener in ähnlicher Form erstellt werden.

```
1 mSeekBar_1.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
2    @Override
3    public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
4       sendCommand("*1|"+String.valueOf(i)+"#");
5    }
6    @Override
7    public void onStartTrackingTouch(SeekBar seekBar) {
```

```
8         sendCommand("*1|"+String.valueOf(seekBar.getProgress())+"#");
9     }
10    @Override
11    public void onStopTrackingTouch(SeekBar seekBar) {
12        sendCommand("*1|"+String.valueOf(seekBar.getProgress())+"#");
13    }
14 });
```

Quellcode 8: Seekbar Listener Beispiel

Dies sind die wichtigsten Programmteile, die die Eingabe von Steuerbefehlen vom Nutzer ermöglichen. Für weitere Programmdetails soll hier auf den kommentierten Programmcode verweisen werden, der zusammen mit dieser Arbeit abgegeben wurde.

4.3.5 Finale Applikation

Das Steuern der Drohne nur mit Hilfe einzelner Schieberegler wäre sehr umständlich und entspricht nicht dem üblichen Design von Fluggerät-Fernsteuerungen. Daher verfügt die finale Handyapplikation über Joysticks, wie sie auch auf Standardfernsteuerungen zu finden sind. Diese sind in der Standard-View-Bibliothek die das Applikationsframework zur Verfügung stellt jedoch nicht vorhanden und mussten erst selbst erstellt werden. Hierbei wurde auf eine möglichst einfache Bedienung Rücksicht genommen, weshalb dem Nutzer nur eine eingeschränkte Steuermöglichkeit angeboten wird. Ein Freiheitsgrad (Yaw) der Steuerung wurde daher entfernt (siehe Abbildung 7).

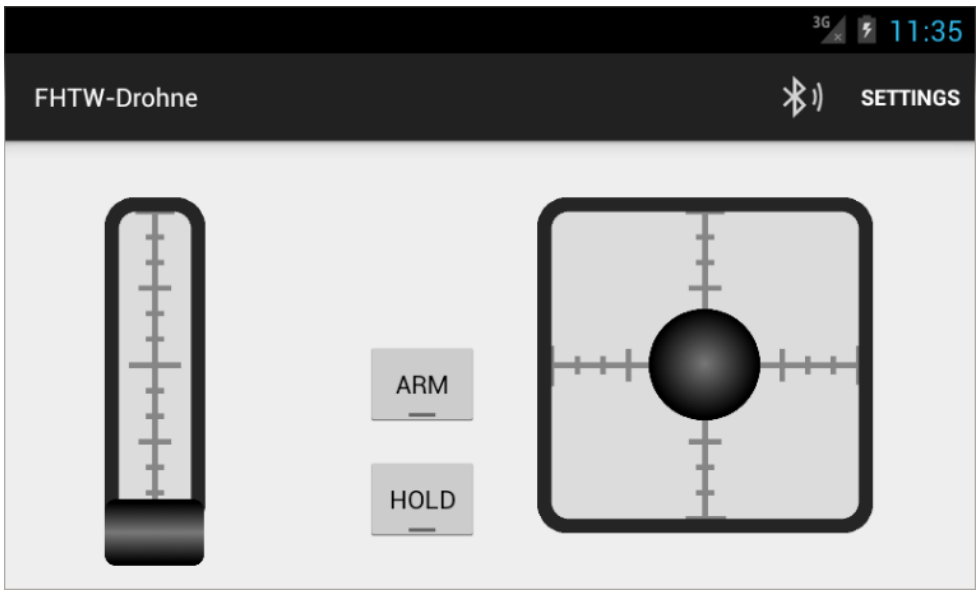

Abbildung 7: Finale Steuerungsapplikation

Des Weiteren wurde ein zusätzliches Settings-Menü erstellt, in dem die Sensitivität der Steuerinput angepasst und Kalibrierungseinstellungen getroffen werden können (siehe Abbil-

dung 8). Im Folgenden sollen erläutert, wie diese Anpassungen umgesetzt wurden und welche Veränderungen dazu im Vergleich zur Testapplikation notwendig waren.

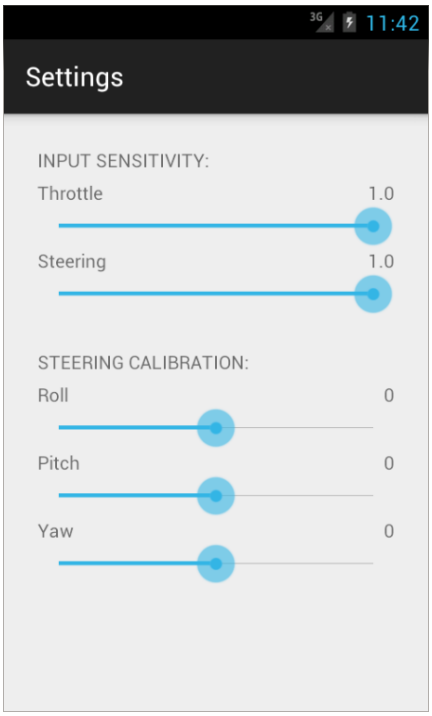

Abbildung 8: Settings der Finalen Applikation

Umsetzung der Anpassungen

Zu aller erst mussten individuelle Views erstellt werden: ein vertikaler Schieberegler zur Steuerung von *Throttle* und ein Joystick zum Einstellen von *Roll* und *Pitch* der Drohne. Dazu wurden Grafiken erstellt, die die Basis der Steuerelemente und die Hebel zeigen. Anschließend wurden zwei neue Klassen erstellt (`ViewSlider` und `ViewJoystick`), die die Basisklasse View erweitern. Alle Maße innerhalb der beiden selbst erstellten Views sind relativ zu dessen Größe angegeben, in der es in die Benutzeroberfläche eingebunden wird. Dadurch bleiben das Größenverhältnisse der Basis zum Steuerhebel immer gleich. In beiden View-Klassen wurde auch eine Listener-Funktion vorgesehen, mit der die aktuelle Stellung des Steuerelements ausgelesen werden kann. Die Ausgabewerte befinden sich hierbei innerhalb der Wertebereiche, die vom Steuerbefehl-Protokoll vorgegeben werden (siehe Tabelle 2 in Abschnitt 4.2.1). Für eine detaillierte Codebeschreibung wird auch hier auf die Codekommentare im den entsprechenden Dateien verweisen (`ViewSlider.java` und `ViewJoystick.java`).

In der Hauptaktivität der Handyapplikation wurden die neuen Views für die Benutzereingabe von *Throttle* (`ViewSlider`) und von *Roll* und *Pitch* (`ViewJoystick`) eingebunden (siehe Quellcode 9; Zeilen 3, 4) und damit die bisher verwendeten SeekBars ersetzt. Dementsprechend mussten auch neue Listener für beide Views erstellt werden. Beispielhaft wird im

Quellcode 9 der Listener des Joysticks gezeigt. Dabei wird einerseits die Joystickstellung (`valueX`, `valueY`) berücksichtigt, andererseits die in den Einstellungen gesetzte Sensitivität und die Offset-Werte zur Kalibrierung (Codezeilen 12 – 16). Schließlich werden die entsprechenden Steuerbefehle über Bluetooth versandt (Zeilen 18, 19). Der zweite Funktionswert der `setOnJoystickMoveListener`-Funktion gibt das Zeitintervall vor mit dem die Steuerbefehle erneut gesendet werden. Voreingestellt sind in der Variable `DEFAULT_LOOP_INTERVAL` $50\ ms$.

```
1 protected void onCreate(Bundle savedInstanceState) {
2    ...
3    mSlider = (ViewSlider)findViewById(R.id.throttleView);
4    mJoystick = (ViewJoystick)findViewById(R.id.joystickView);
5    ...
6 }
7 private void setupControl() {
8
9 mJoystick.setOnJoystickMoveListener(new OnJoystickMoveListener() {
10    @Override
11    public void onValueChanged(float valueX, float valueY) {
12       float sens = mSharedPreferences.getFloat(mSSens, 1);
13       int roll_cal = mSharedPreferences.getInt(mRoll, 0);
14       int pitch_cal = mSharedPreferences.getInt(mPitch, 0);
15       int roll = Math.round(valueX*sens)+roll_cal;
16       int pitch = Math.round(valueY*sens)+pitch_cal;
17
18       sendCommand("*2|" + String.valueOf(roll) + "#");
19       sendCommand("*3|"+String.valueOf(pitch)+"#");
20    }
21 }, ViewJoystick.DEFAULT_LOOP_INTERVAL);
```

Quellcode 9: Slider Listener der finalen Applikation

Zu guter Letzt wurde eine zusätzlicher Menüpunkt verwirklicht. Dieser erlaubt es die Sensitivität der Steuerinputs fest zu legen. Dabei können die Empfindlichkeiten für *Trottle* und den Richtungssteuerbefehlen *Roll* und *Pitch* unabhängig von einander festgelegt werden. Des Weiteren können die Richtungsbefehle kalibriert werden. Falls die Drohne also in neutraler Joystickstellung die Tendenz zeigt sich zu drehen, oder in eine Richtung zu driften können hier Offsets im Bereich von -100 bis +100 eingestellt werden um diese Tendenzen auszugleichen. Innerhalb der Applikation wurde dies mit `SharedPreferences` gelöst. Mit Hilfe dieses Interfaces können selbst deklarierte Einstellungen gespeichert und editiert werden. Damit können globale Variablen nicht nur aktivitätsübergreifend (von Aktivität zu Aktivität) genutzt, sondern auch von Programmstart zu Programmstart langfristig gespeichert werden. Quellcode 10 zeigt die notwendigen Codezeilen zum Erstellen (Zeile 10), Auslesen (Zeile 13) und Beschreiben (Zeilen 18 – 20) von SharedPreferences am Beispiel der Einstellung der Throttle-Sensitivität (SeekBar `m_SeekTSens`).

```
1 // Shared Preferences for global variables
2 SharedPreferences mSharedPreferences;
3 public static final String myPREFERENCES = "PrefSettings";
4 public static final String mTSens = "Throttle_Sensitivity";
5 public static final String mSSens = "Steering_Sensitivity";
6 public static final String mRoll = "RollCal";
7 public static final String mPitch = "PitchCal";
8 public static final String mYaw = "YawCal";
9
10 mSharedPreferences = getSharedPreferences(myPREFERENCES, Context.MODE_PRIVATE);
11 m_SeekTSens = (SeekBar)findViewById(R.id.seek_throttle_sens);
12
13 m_SeekTSens.setProgress((int)(mSharedPreferences.getFloat(mTSens, 1)*1000));
14
15 m_SeekTSens.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
16    @Override
17    public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
18       SharedPreferences.Editor editor = mSharedPreferences.edit();
19       editor.putFloat(mTSens, (float) i / 1000);
20       editor.commit();
21    }
22    ...
23 }
```

Quellcode 10: SharedPreferences der finalen Applikation

Mit all den hier beschriebenen Änderungen konnte die finale Applikation erstellt werden, die eine flexible und benutzerfreundliche Drohnensteuerung zulässt.

5 Resultate

Mit Hilfe der im vorangegangen Kapitel beschriebenen Systemarchitektur ist es im Zuge dieses Projekts gelungen eine Mini-Drohne – sie passt bequem auf eine Handfläche – zu erstellen, die mit Hilfe einer Smartphone-Applikation steuerbar ist (siehe Abbildung 9).

Bei den durchgeführten Tests konnte eine Funkreichweite, innerhalb derer ein sicherer Datentransport sichergestellt ist, von 10 bis $20\ m$ festgestellt werden. Der verbaute FC stabilisiert das Fluggerät automatisch, was das Steuern der Drohne sogar Neulingen im Bereich des Modellfluges ermöglicht. Ein gutes Beispiel hierfür ist der Autor dieser Arbeit und sein Projektkollege Herr Otrebski, denen ohne jegliche vorangegangen Flugerfahrung das Lenken der Drohne gelungen ist. Um das Driften der Drohnenposition im Flug ohne Steuerinput zu minimieren und somit auch das Lenken derselben zu erleichtern muss die Möglichkeit zur Kalibrierung genutzt werden, welche durch die Applikation gegeben ist.

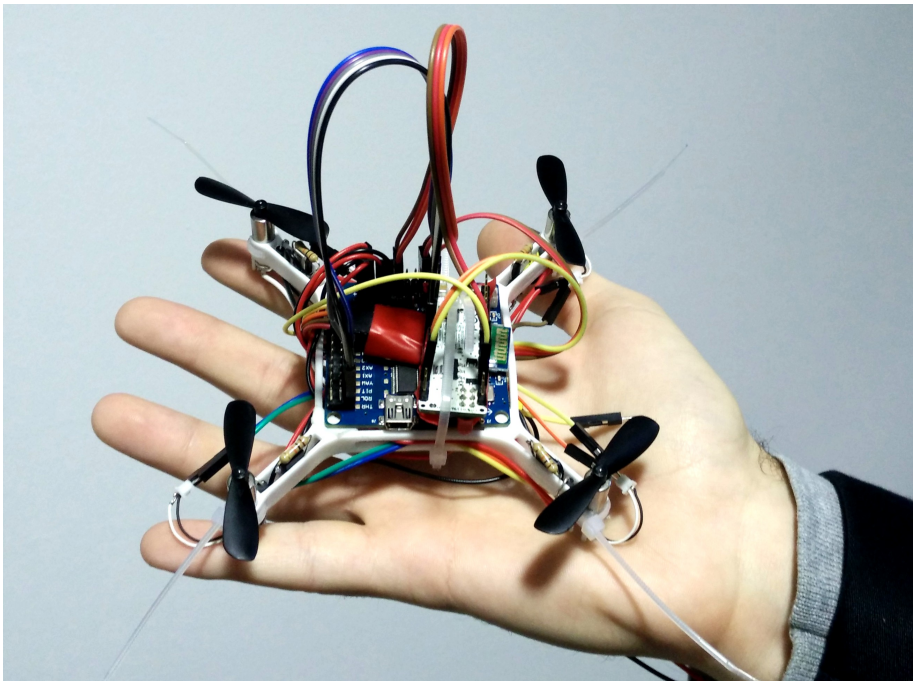

Abbildung 9: Erster Prototyp der FHTW-Drohne

Innerhalb dieses Projektes wurde die Funktionsfähigkeit Drohne unter Laborbedingungen (angeschlossen an ein Netzgerät, ohne eigenen Akku getestet. Der Aktionsradius war daher stark eingeschränkt. Das zusätzliche Gewicht, mit dem durch den Akku gerechnet werden kann sollte kein Problem darstellen. Selbst bei erst ca. 50 % der möglichen Antriebsleistung der Motoren konnte die Drohne abheben.

Prinzipiell ist die hier vorgestellte Systemarchitektur für Drohnen aller Größen und Bauarten einsetzbar. Die grundlegenden Steuerelemente (FC, Steuer-Mikrocontroller und Bluetooth-Shield) bleiben hierbei die selben, es ändert sich lediglich die Konfiguration und/oder Dimension der Drohne. So können die im Laufe dieses Projektes entstandenen Programme jederzeit auch für andere Multicopter-Drohnen weiterverwendet werden.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde kurz der Stand der Technik im Bereich der Ferngesteuerten Klein-Drohnen gezeigt. Anschließend wurde eine Systemarchitektur und die damit verknüpfte Programmierung vorgestellt, mit der ein durch eine Handy-Applikation gelenkter Multicopter erfolgreich gesteuert werden konnte. Dabei wurde großer Wert auf Modularität Wert gelegt. Dies ermöglicht den einfachen Austausch einzelner Bauteile durch Komponenten mit ähnlichem Funktionsumfang. Des Weiteren sollte dadurch ein starker Eingriff in die Funktion einzelner Teile

– zB dem FC – vorgebeugt werden und das Erweitern des Systems – zB durch zusätzliche Intelligenz im Steuer-Mikrocontroller – ermöglicht werden.

In zukünftigen Projekten können daher einerseits Individualintelligenz sehr einfach am Steuer-Mikrocontroller hinzugefügt werden. Hier bieten sich verschiedene Funktionen an, die einfach von der Handy-Applikation aus gestartet werden können. Die Umsetzung einer automatischen Landung, oder das Halten der aktuellen Position wären hier sehr nützlich. In weiterer Folge kann das System auch in Richtung der zu Beginn dieser Arbeit angesprochenen Schwarmintelligenz erweitert werden. Dies wäre eine Möglichkeit erste Schritte in das Forschungsfeld der Schwarmrobotik zu wagen.

Auch die Smartphone-Applikation könnte erweitert werden. Hier bietet sich das Ausnutzen der internen Sensorik des Telefons an. Dadurch könnte beispielsweise eine Steuerung der Drohne entsprechend der Orientierung des Mobiltelefons realisiert werden. Um fortgeschrittenen Usern alle Freiheitsgrade der Steuerung zur Verfügung zu stellen (die Änderung von *Yaw* wurde in der aktuellen Applikation verhindert) könnte in den Einstellungen die Wahl eines Expertenmodus angeboten werden, der dem Benutzer alle Steuermöglichkeiten frei gibt.

Bevor jedoch all dies umgesetzt werden kann, ist das Auslegen und Hinzufügen eines Akkus notwendig. Durch ein Redesign der Drohne, die bisher noch im Prototypen-Stadium ist, können auch die Bauteile platzsparender und übersichtlicher angeordnet werden. Dies würde schließlich auch zu einer Reduktion der Kabellängen – für interne Verbindungen – und somit zu einer Verringerung des Gewichte der Drohne führen.

Literaturverzeichnis

Amazon Europe Core S.à r.l., 2015. *Hubsan X4 Drone*. [Online] Verfügbar unter: <http://www.amazon.de/Hubsan-X4-Drone-Quadrocopter-Batterien/dp/B009M1PO7W/ref=sr_1_14?ie=UTF8&qid=1421699830&sr=8-14&keywords=hubsan> [Zugang am 19.01.2015].

Arduino, 2015*a*. *Arduino Micro*. [Online] Verfügbar unter: <http://arduino.cc/en/Main/ArduinoBoardMicro> [Zugang am 17.01.2015].

Arduino, 2015*b*. *Servo library*. [Online] Verfügbar unter: <http://arduino.cc/en/reference/Servo> [Zugang am 17.01.2015].

Arduino, 2015*c*. *SoftwareSerial Library*. [Online] Verfügbar unter: <http://arduino.cc/en/Reference/softwareSerial> [Zugang am 17.01.2015].

Ars Electronica Linz GmbH, 2014. *Spaxels*. [Online] Verfügbar unter: <http://www.aec.at/spaxels/> [Zugang am 20.10.2014].

Bake, P., 2015. *Flyduino - smooth flying since 2011*. [Online] Verfügbar unter: <https://flyduino.net> [Zugang am 19.01.2015].

campilots GmbH, 2014. *campilots - Mini Helicopter Filmaufnahmen*. [Online] Verfügbar unter: <http://www.campilots.com/de/home.html> [Zugang am 20.10.2014].

DIGICOPTER® Luftaufnahmen & Videoproduktion, 2014. *DIGICOPTER.DE*. [Online] Verfügbar unter: <http://digicopter.de/de/luftaufnahmen/> [Zugang am 20.10.2014].

Dubus, A., 2015. *MultiWii*. [Online] Verfügbar unter: <http://www.multiwii.com/> [Zugang am 20.01.2015].

Guangzhou HC Information Technology Co., Ltd., 2014. *Product Data Sheet - HC-06*. [Online] Guangzhou, China: Guangzhou HC Information Technology Co., Ltd. Verfügbar unter: <http://silabs.org.ua/bc4/hc06.pdf> [Zugang am 17.01.2015].

hobbyking.com®, 2015*a*. *Hobbyking KK2.0 Multi-rotor LCD Flight Control Board*. [Online] Verfügbar unter: <http://www.hobbyking.com/hobbyking/store/__34542__Hobbyking_KK2_0_Multi_rotor_LCD_Flight_Control_Board_EU_warehouse_.html> [Zugang am 20.01.2015].

hobbyking.com®, 2015*b*. *MultiWii 328P Flight Controller*. [Online] Verfügbar unter: <http://www.hobbyking.com/hobbyking/store/__27033__MultiWii_328P_Flight_Controller_w_FTDI_DSM2_Port.html> [Zugang am 20.01.2015].

Künneth, T., 2012. *Android 4: Apps entwickeln mit dem Android SDK*. Bonn: Galileo Press.

PJRC.COM, L., 2015. *AltSoftSerial Library*. [Online] Verfügbar unter: <http://www.pjrc.com/teensy/td_libs_AltSoftSerial.html> [Zugang am 17.01.2015].

Shenzhen Hubsan Technology Company Limited, 2015. *Hubsan X4 Cemare*. [Online] Verfügbar unter: <http://www.hubsan.com/productinfo_16.html> [Zugang am 17.01.2015].

Tutorials Point, 2015. *Android Programming Tutorial*. [Online] Verfügbar unter: <http://www.tutorialspoint.com/android/> [Zugang am 19.01.2015].

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

bps Bits pro Sekunde

FC Flight-Controller

I²C Inter-Integrated Circuit

MSP MultiWii Serial Protocol

PWM Pulsweitenmodulation

UART Universal Asynchronous Receiver Transmitter

UAV Unmanned Aerial Vehicle