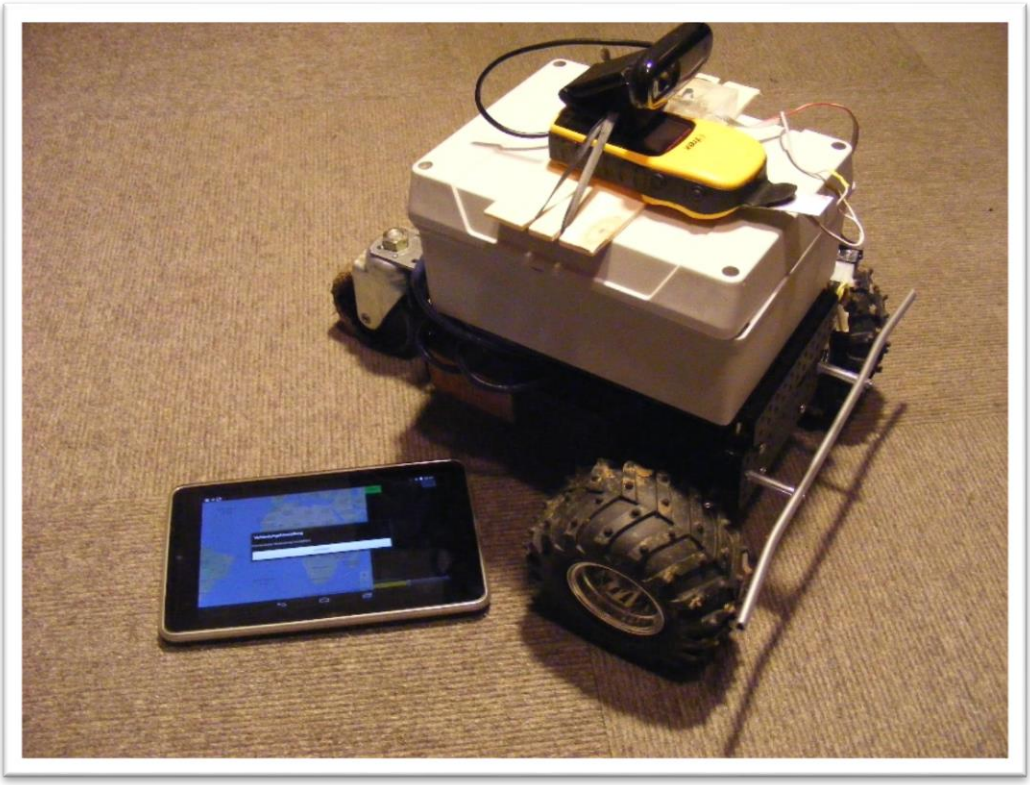
Matthias Bernard und Sebastian Knopp

SmartBot

Ein Roboter mit Android und Linux



14.1.2014

Inhalt

1 Vorgeschichte

Abbildung 1: Raspberry Pi

Anfang der Osterferien 2013 bestellten wir uns einen „Raspberry Pi“ im Internet, anfangs ohne große Pläne, was damit wohl anzufangen sei. Der „Raspberry Pi“ ist ein kreditkartengroßer Einplatinencomputer, der von der Raspberry Pi Foundation entwickelt wurde. Er ist seit Anfang des Jahres 2013 in Deutschland verfügbar und erfreut sich im Internet großer Beliebtheit. Für 40€ bekommt man einen ARM SoC (System-on-a-Chip) mit Netzwerk, USB, vielen weiteren Schnittstellen, einer angepassten Debian Linux Distribution und einem Stromverbrauch von 3,5 Watt an 5V. Nachdem der kleine Rechner auf dem Schreibtisch lag, stellte man sich natürlich die Frage, was sich mit dem Stück Hardware anfangen lässt. Das Internet war zum Glück reich an Beispielanwendungen und Anleitungen. Nach einer Woche Experimentieren konnte unser Raspberry Pi:

- einen Webserver hosten
- eine Schreibtischlampe über das Internet eine und ausschalten
- als WLAN Zugangspunkt agieren
- das Bild einer angeschlossenen USB Webcam über das Netzwerk streamen

Bald kam die Idee auf, all diese Funktionalitäten zu kombinieren und einen Roboter zu bauen, der sich über das Netzwerk steuern lässt, und das Livebild der Webcam aus der Perspektive des Roboters überträgt. Die erste Version war schnell funktionstüchtig und bestand aus einer Website, die das Bild der Webcam und Buttons zur Ansteuerung der zwei Lego Elektor Motoren enthielt. Mit diesem Aufbau drehte unser Roboter seine erste Runde durch das Wohnzimmer. Probleme bereiteten uns vor allem hohe Latenzen und die Unzuverlässigkeit, die auf die fehlende Eignung von PHP für diesen Aufgabenbereich zurückzuführen ist.

Angespornt von diesem ersten Erfolg entschlossen wir uns, uns mit dem Thema Roboter etwas ausführlicher zu beschäftigen, wobei der Raspberry Pi weiterhin die Basis für unsere Versuche bilden sollte.

2 Planung

Im Rahmen unseres Projekts entwickeln wir also einen Roboter, der sich zunächst über eine kleine Java-Anwendung via WLAN steuern lassen sollte. Der Roboter wurde mit einer geländegängigen Motorplattform, einer selbst gebauten Motorsteuerung, einer Webcam, einem WLAN-Stick und dem Raspberry Pi ausgestattet. Zu diesem Stand luden wir ein Video auf YouTube hoch.[1] Da uns ein Laptop zur Steuerung auf Dauer zu unhandlich erschien, entschieden wir uns für die Umsetzung der Steuerung über eine Android-App. Zusätzlich zur manuellen Steuerung sollte der Roboter auch in der Lage sein, anhand von GPS-Koordinaten selbstständig ein Ziel anzusteuern. Wir mussten den Roboter hierzu zusätzlich mit einem Kompass und einem GPS-Empfänger ausstatten und in der App einen zusätzlichen Menüpunkt einplanen.

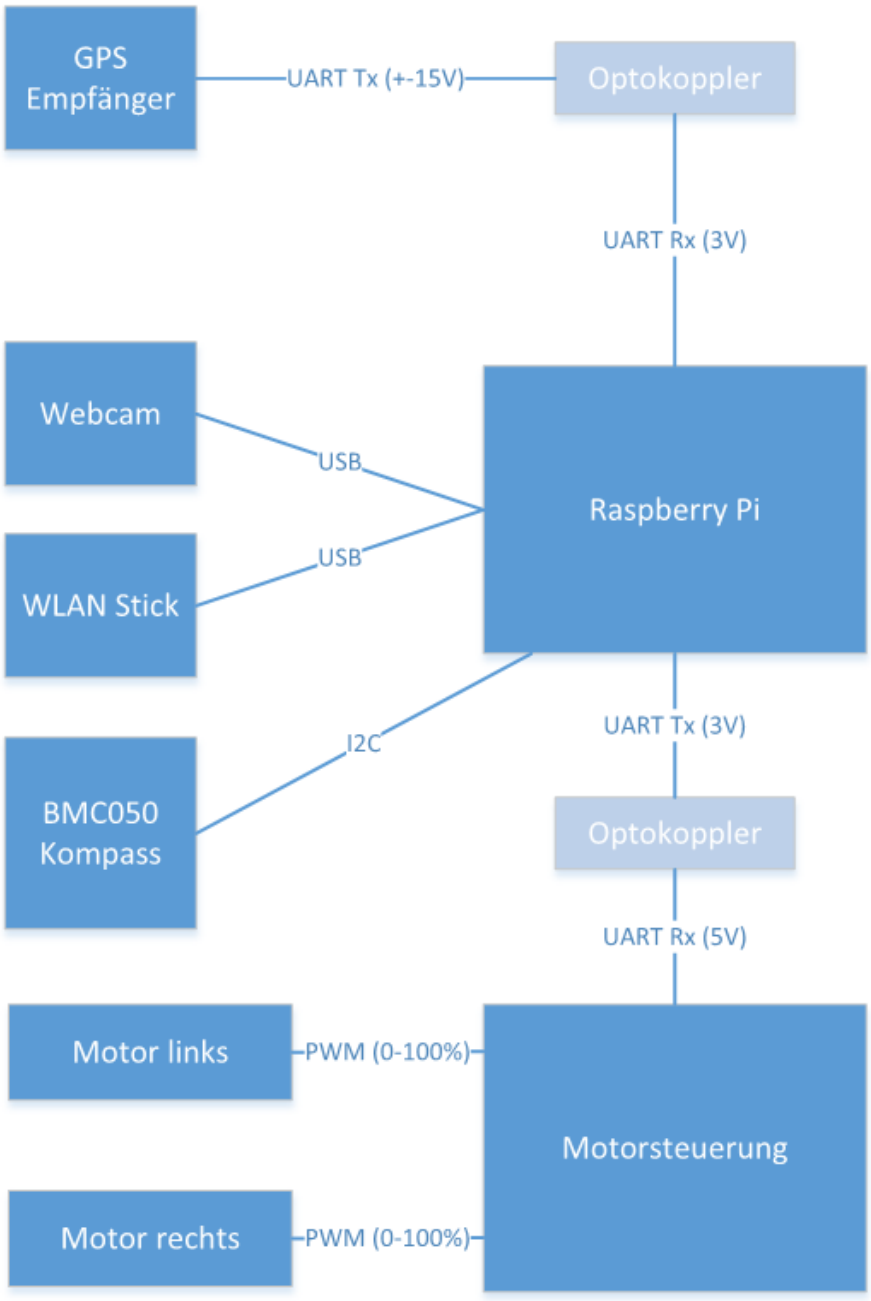


Abbildung 2: Schematische Darstellung des Roboters

[1] **Video zum damaligen Entwicklungsstand (20.05.2013):** http://youtu.be/vJV69kW2TUc

3 Aufbau

3.1 Motorplattform

Als fahrbaren Untersatz für unseren Roboter dient ein Metallchassis, an dem sich vorn zwei Getriebemotoren mit großen Rädern und hinten ein drehbar gelagertes Hartplastikrad befinden. Die Lenkung erfolgt über die Drehzahldifferenz zwischen den Elektromotoren der Vorderräder.

3.2 Motorsteuerung

Um den Roboter steuern zu können muss sich die Drehrichtung und Geschwindigkeit der Motoren über den Raspberry Pi steuern lassen. Die Motoren benötigen jeweils eine Spannung von maximal 7V und einen maximalen Strom von 5 Ampere. Um eine Drehung in beide Richtungen und eine stufenlose Geschwindigkeitsregelung zu ermöglichen, muss also ein Spannungsbereich von -7V bis +7V abgedeckt werden. Die Datenpins des Raspberry Pi liefern lediglich einen Pegel von 3,3V bei einer Belastbarkeit von einigen Milliampere. Eine direkte Ansteuerung der Motoren ist also ausgeschlossen. Das Signal des Raspberry Pi muss also mittels eines Transistors verstärkt werden, um die für den Motor nötige Spannungs- und Strombelastbarkeit zu gewährleisten. Mittels eines einzelnen Transistors ist es allerdings noch nicht möglich, den Motor in beide Richtungen zu steuern, also vorwärts und rückwärts drehen zu lassen. Diese Funktionalität wird mit Hilfe einer Brückenschaltung aus vier Transistoren realisiert. Diese ermöglicht es, den angeschlossenen Motor im Betrieb umzupolen, und so die Laufrichtung zu ändern. Zuletzt muss man eine Lösung finden, mit der man die Geschwindigkeit der Motoren am besten regeln kann. Wir überlegten, die Transistoren mit einer regelbaren Spannung anzusteuern und sie so, je nach gewünschter Geschwindigkeit teilweise oder komplett durchschalten zu lassen. Dieses Verfahren hat jedoch den Nachteil, dass ein großer Teil der Energie in Wärme umgewandelt wird, außerdem fehlt dem Raspberry Pi der nötige analoge Ausgang. Stattdessen haben wir uns für die so genannte „Pulsweitenmodulation“ kurz PWM entschieden. Hierbei wird der Motor mit einer pulsierenden Gleichspannung konstanter Frequenz angesteuert (in unserem Fall liegt die Frequenz bei einigen Kilohertz). Die Leistung, also in unserem Fall die Geschwindigkeit, wird über den Tastgrad bestimmt. Der Tastgrad beschreibt das Verhältnis zwischen der Zeit, in dem ein High-Signal am Motor anliegt und der Periodendauer. In der Grafik (siehe oben rechts) beträgt der Tastgrad demnach 25%. Stark vereinfacht ausgedrückt läuft der Motor mit 25% seiner maximalen Leistung bzw. Geschwindigkeit. So entstehen minimale Verluste und man benötigt für die Generierung des PWM-Signals nur einen einfachen, digitalen Ausgang.

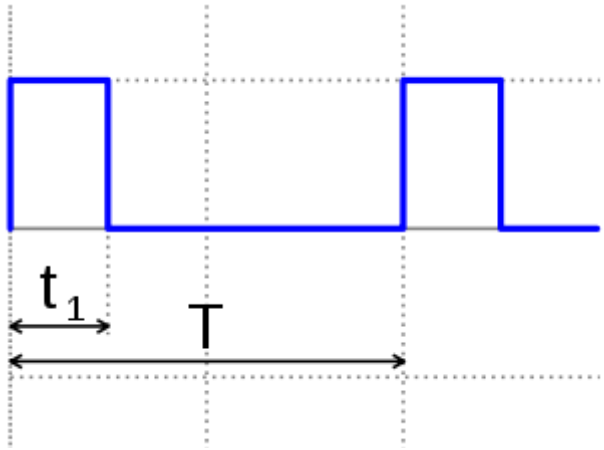


Abbildung 3: Pulsweitenmodulation

4 Umsetzung

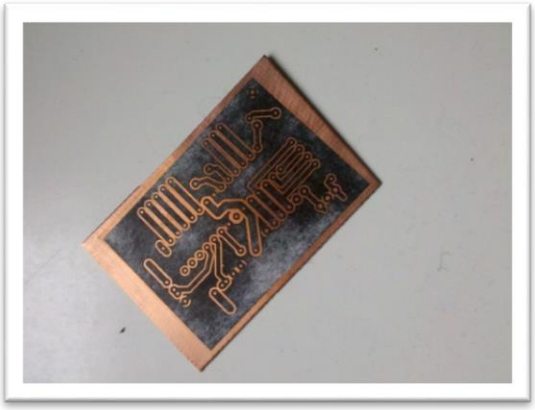


Abbildung 4: Tonerbeschichtete Platine vor dem Ätzen

Bei der Gestaltung der Schaltung haben wir uns dafür entschieden, die PWM-Signale für die zwei Brückenschaltungen von einem Mikrocontroller generieren zu lassen. Zum einen ist Linux wenig für Echtzeitaufgaben, wie das Generieren von PWM-Signalen geeignet, zum anderen erleichtert es die elektrische Trennung zwischen dem Stromkreis der Motoren und dem Raspberry Pi, um im Fehlerfall eine Beschädigung vorzubeugen. Als Controller nutzen wir einen „Atmega 8“ der Firma Atmel. Diesen programmieren wir in C, um Befehle vom Raspberry Pi über die serielle Schnittstelle zu empfangen und ein entsprechendes PWM-Signal für Brückenschaltung des linken bzw. rechten Motors zu generieren. Über die serielle Schnittstelle kann der Raspberry Pi Werte zwischen +127 und -127 für den linken bzw. rechten Motor setzen (positiver Wert = vorwärts, negativer Wert = rückwärts). Die zwei Brückenschaltungen bestehen aus je vier MOSFETs mit einer Strombelastbarkeit von mehr als 10A. MOSFETs haben gegenüber den klassischen Transistoren den Vorteil einer geringeren Verlustleistung im geschalteten Zustand, außerdem benötigen sie einen geringeren Steuerstrom.

Die Schaltung bauen wir auf selbstgeätzten Platinen auf. Die Layouts für die Platinen werden mit einem speziellen Programm am PC gezeichnet und mit einem Laserdrucker auf Hochglanzpapier ausgedruckt. Das ausgedruckte Platinenlayout wird mittels Bügeleisen auf eine Kupferbeschichtete Platine aufgebügelt. Auf der Platine sind jetzt die Bereiche, an denen sich später die Leiterbahnen befinden sollen mit Toner bedeckt (siehe Grafik).Bei dem eigentlichen Ätzvorgang in einer Eisen(III)-chlorid Lösung wird das Kupfer an den blanken Stellen weggeätzt und übrig bleiben am Ende die mit Toner bedeckten Leiterbahnen. Nach dem Bohren der Löcher kann die Platine mit den Bauteilen bestückt werden. Die serielle Datenleitung zwischen Raspberry Pi und dem Mikrocontroller wird über einen Optokoppler verbunden. Dieser gewährleistet eine galvanische Trennung und eine Anpassung des Spannungspegels zwischen Mikrokontroller (5V) und Raspberry Pi (3,3V).

4.1 GPS

Seine eigene Position ermittelt der Roboter mittels GPS. Als Empfänger dient ein GPS-Handgerät der Marke „Garmin“, welches über eine serielle Schnittstelle verfügt. Über diese Schnittstelle wird mit einem Intervall von 1 Hz die aktuelle Position in Form von geographischen Koordinaten ausgegeben. Die Schnittstelle entspricht dem RS232 Standard mit einem Pegel von -10V bis +10V. Die Anpassung an den 3,3V Pegel des Raspberry Pi erfolgt auch an dieser Stelle mit einem Optokoppler. Die Klasse „Gps.java“ übernimmt die Auswertung der über die serielle Schnittstelle eingehenden Daten. Wird ein Datensatz mit neuen Koordinaten erkannt, werden die Koordinaten geparst und in zwei Variablen von Typ „double“ abgespeichert. Über diese Koordinaten kann nun im Rest des Programms die Position des Roboters verarbeitet werden, z.B. um die Entfernung bis zu einem festgelegten Zielpunkt zu berechnen. Jedes Mal, wenn der GPS Empfänger die Position über die serielle

Schnittstelle sendet, werden die Variablen, welche Koordinaten enthalten aktualisiert. Außerdem wird anhand der Koordinaten und der Zielkoordinaten der Zielkurs berechnet. [2]

4.2 Kompass

Zusätzlich zur eigenen geographischen Position benötigt der Roboter Information über den Kurs, also in welche Himmelsrichtung er fährt. Theoretisch lässt sich der Kurs aus der aktuellen Position und der vorletzten erfassten Position errechnen. In der Praxis hat sich dieses Verfahren jedoch nicht bewährt, weil es, bedingt durch die beschränkte Genauigkeit von GPS, zu sehr ungenauen Ergebnissen kommt. Aus diesem Grund haben wir uns für die Verwendung eines digitalen Kompasses entschieden, mit dem sich der Kurs des Roboters in Grad (0°-360°) präzise bestimmen lässt. Nach einiger Recherche und einer E-Mail an „Bosch Sensortec“ erhielten wir kostenlos einen digitalen Kompass vom Typ BMC050.

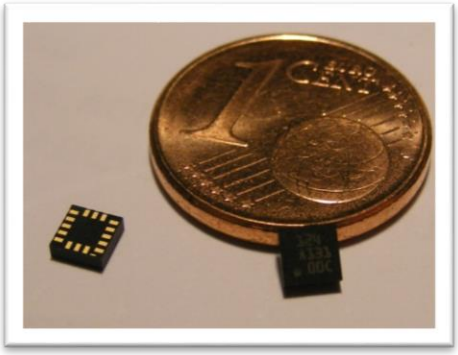


Abbildung 5: Kompass im Größenvergleich

Der BMC050 selbst ist ein 3mm x 3mm x 0.95mm großer Chip mit 16 Kontaktpads. Die Bauform des Chips ist eigentlich für ein „surface mount“, also das direkte Auflöten der Kontaktpads auf die Leiterbahnen gedacht, jedoch lassen sich mit den uns zur Verfügung stehenden Möglichkeiten keine so feinen Leiterbahnen herstellen. Stattdessen löten wir an die Kontaktpads sehr dünnen Kupferdraht, über den der Chip mit einer selbst geätzten Platine verbunden ist. An externer Beschaltung benötigt der Chip lediglich einen kleinen Kondensator parallel zur Spannungsversorgung. Die Anbindung an den Raspberry Pi erfolgt über den I2C-Bus.
Eine Anpassung des Pegels ist an dieser Stelle nicht nötig. So kann der Kompass direkt an den Raspberry Pi angeschlossen werden. Nach Anschluss des Kompasses finden sich am I2C-Bus zwei neue Geräteadressen. Die eine Adresse zeigt auf ein Accelerometer, das ebenfalls im Chip integriert ist, von uns aber nicht benötigt wird. Die andere Adresse zeigt auf den Kompass. Über diese Adresse kann auf die bis zu 255 8-Bit-Register des Kompasses zugegriffen werden. Im Grunde besteht der Kompass aus drei Magnetfeldsensoren, die die Stärke des Erdmagnetfelds in x-, y- und z-Richtung erfassen. Interessant sind für unseren Zweck nur die Messwerte der x- und y-Achse. Die Messwerte werden mit einer Auflösung von 10 Bit erfasst und in zwei 8-Bit-Registern gespeichert. Bevor man jedoch die Messwerte der x- und y-Achse aus den Registern auslesen kann, muss der Kompass erst einmal aktiviert und konfiguriert werden. Aus dem Datenblatt geht hervor, dass der Kompass über das Schreiben eines Bits in das passende Register aktiviert werden kann. Über ein weiteres Register kann die Frequenz eingestellt werden, mit der neue Messwerte in die Ausgaberegister geschrieben werden. Für unsere Zwecke ist eine Wiederholfrequenz von 30Hz ausreichend. Nach dem einmaligen Schreiben der Konfigurationsregister liest die Java Kompass-Klasse in einer Schleife immer weiter die aktuellen Messwerte aus den entsprechenden Registern für den x- und y-Sensor aus. Sie berechnet

[2] **Berechnung des Kurses zwischen zwei Koordinaten**:
http://www.smokycogs.com/blog/tag/geodesy/

anhand der Messergebnisse den aktuellen Kurs[3] und speichert ihn in der Variable „kurs“ vom Typ „double“ ab.

Die rohen Messwerte sind vorzeichenbehaftete binäre Werte, daher ist eine Umrechnung nötig. Das Vorzeichen wir durch das Bit mit der höchsten Wertigkeit (bei einem 8-Bit-Wert 2^7) angegeben. Hat diese Bit den Wert „1“, so ist das Vorzeichen negativ und die restlichen Bits sind invertiert. Ist der Wert hingegen „0“, so ist das Vorzeichen positiv und es bedarf keiner weiteren Umrechnung.

Beispiel für die Umrechnung eines vorzeichenbehafteten 8Bit Werts in einen vorzeichenbehafteten Dezimalwert.

Binär	Dezimal
10000110	134

Der Wert ist größer als 2^7, demnach handelt es sich um einen Wert mit negativem Vorzeichen. Zum Aufheben der Invertierung rechnet man $134 - (2^8 - 1)$ (2^8weil es sich um einen 8Bit Wert handelt). Somit repräsentiert 0b10000110 in vorzeichenbehafteter binärer Darstellung den Dezimalwert -121.

4.3 WLAN und Webcam

Die verwendete Kamera ist eine Standard USB-Webcam. Zur Übertragung des Webcam-Bildes nutzen wir einen „Motion JPEG“(MJPEG) Stream der über „HTTP“ an die Android-App übertragen wird. Beim MJPEG Video-Codec wird jedes Frame als JPEG-Bild gespeichert und übertragen. Die Webcam unterstützt Hardware-Codierung dieses Formats, wodurch die Belastung für den Raspberry Pi gering bleibt. Nachteil ist der hohe Bandbreitenbedarf, weil für jedes Frame ein vollständiges Bild übertragen werden muss. Daher ist ein flüssiger Video-Stream nur mit geringer Videoauflösung möglich.

Für die WLAN-Verbindung haben wir einen WLAN-Stick an den Raspberry Pi angeschlossen. Die Software „hostapd“ lässt den Raspberry Pi als WLAN-Zugangspunkt mit WPA-Verschlüsselung agieren. Über WLAN können sich nun Smartphones direkt mit dem Roboter verbinden. Der DHCP-Dienst „dhcpd“ verteilt an verbundene Geräte eine IP aus dem 192.168.1.0er-Netz. Der Raspberry Pi selbst trägt an der WLAN-Schnittstelle die statische Adresse 192.168.1.50. Über die Ethernet-Schnittstelle lässt sich der Roboter weiterhin über LAN ins Heimnetzwerk einbinden.

4.4 Programmierung

Bei der Programmierung haben wir uns für die Programmiersprache Java entschieden. Sowohl Android-Apps, als auch Anwendungen für den Raspberry Pi lassen sich in Java programmieren. Dadurch ersparen wir uns den ständigen Wechsel zwischen zwei Programmiersprachen. Weiterhin können auf diese Weise Codeblöcke, die z.B. für den Server geschrieben wurden, ohne weiteres auch in der Android-App genutzt werden. Der Zugriff auf die Hardwareschnittstellen des Raspberry Pi, wie z.B. die serielle Schnittstelle oder der I2C-Bus unter Java wird durch die Bibliothek „Pi4J“ ermöglicht.[4] Unglücklicherweise fehlte in der Klasse, die den Zugriff auf die serielle Schnittstelle ermöglicht genau die Datenrate (4800 Bit/s), die wir für das Auslesen des GPS-Empfängers benötigten. Nach einigem

[3] **Berechnung des Kurses anhand der Magnetometer-Messwerte:** http://breakoutjs.com/docs/files/src_io_MagnetoMeterHMC5883.js.html

[4] **„Pi4J“-Bibliothek:** http://pi4j.com/

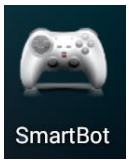
Stöbern im Quellcode der Bibliothek wurde klar, dass die Entwickler lediglich vergessen haben, die von uns benötigte Datenrate in den Definitionsbereich aufzunehmen. Nach Hinzufügen der fehlenden Programmzeile und einem erneuten Kompilieren der Bibliothek konnten wir schließlich die Daten des GPS-Empfängers in unserer Anwendung auswerten.

5 Weboberfläche

Um zum Starten des SmartBot-Servers und des Webcam-Streams nicht immer eine SSH-Verbindung aufbauen zu müssen, haben wir ein simples Webinterface geschrieben, über welches sich der Server und der Webcam-Stream starten und beenden lassen. Die Website wurde in PHP und HTML erstellt. Als Webserver läuft „lighttpd“ mit einer Erweiterung für PHP.

6 Beschreibung der App

Die App stellt das Hauptinstrument zur Steuerung des Roboters dar. Startet man sie, so zeigt das Hauptmenü drei Menüpunkte „Fahrt beginnen“, „Navigation“ und „Einstellungen“.



6.1 Fahrt beginnen

Wählt man „Fahrt beginnen“ aus so öffnet sich eine Android-Activity, welche zwei Joysticks und einen Video-Stream im Hintergrund zeigt. Die Joysticks selber entstammen dem Projekt „IP-Gamepad“[5], wurden für unsere Bedürfnisse aber stark modifiziert. Übernommen wurden sowohl der Netzwerkteil, welches über UDP an einen in den Einstellungen festlegbaren Port Pakete mit der Position der Steuerknüppel im Koordinatensystem der Joysticks sendet, sowie sämtliche Listener für Benutzereingaben und Grafiken. Angepasst wurde die Skalierung und Positionierung der Joysticks und der Wertebereich des Koordinatensystems mit -150 bis +150 auf der x-Achse und -127 bis +127 auf der y-Achse (vgl. Motorsteuerung). Der x-Wert wurde nach einigem Rumprobieren mit der Reaktionsempfindlichkeit des Roboters festgelegt.

Abbildung 6: Ansicht der App im Menüpunkt "Fahrt beginnen"

Der Video-Stream wurde eigenständig in der Activity ergänzt. Er ruft den unter WLAN und Webcam bereits beschriebenen MJPEG-Stream auf und gibt ihn wieder. Um den Stream aufrufen zu können, mussten wir uns längere Zeit mit verschiedenen Protokollen und Codecs auseinandersetzen, da zum einen durch den Raspberry Pi Grenzen geschaffen wurden, zusätzlich aber auch durch Android selbst. Letztlich setzten wir MJPEG ein, obwohl es von Android offiziell nicht unterstützt wird, da dies die einfachste Lösung bot und für unsere Zwecke ausreichend ist.

[5] **IP-Gamepad:** Open-Source Projekt, zu finden unter: https://github.com/ericbarch/IPGamepad

6.2 Navigation

Die Navigations-Activity arbeitet in zwei Threads (Threads = parallel laufende Prozesse), von denen der Hauptthread die Darstellung der Benutzeroberfläche und die Verarbeitung von Benutzerinteraktionen übernimmt, und der zweite Thread eine Netzwerkkommunikation mit dem SmartBot-Server via TCP-Socket ausführt.

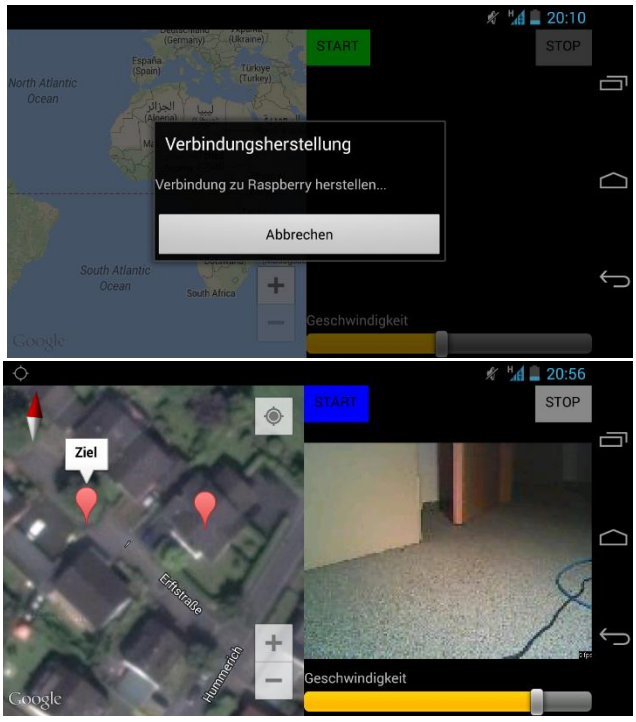


Abbildung 7: Ansicht der App im Menüpunkt "Navigation"

Wird der Menüpunkt gestartet erscheint zunächst ein Popup, welches anzeigt, dass auf eine hergestellte Verbindung gewartet wird. Vom Netzwerkthread wird dann mittels „synchronized-Methode“ das Popup entweder nach einem Timeout auf „keine Verbindung herstellbar“ aktualisiert, oder im Falle eines Erfolges ausgeblendet.
Einmal verschwunden, zeigt die Activity in der linken Bildschirmhälfte eine Google-Maps-Karte, welche sich auf den eigenen Standort zentriert. Außerdem wird eine Stecknadel mit der Roboterposition angezeigt, die sich in einem einstellbaren Intervall aktualisiert. Die rechte Hälfte zeigt oben einen Start- und einen Stopp-Knopf, welche die automatische Navigation starten bzw. beenden. Darunter sieht man erneut den Webcam-Stream, welcher genauso wie in „Fahrt beginnen“ umgesetzt ist, nur diesmal kleiner skaliert wurde. Ganz unten findet sich ein Slider, mit welchem man die Geschwindigkeit des Roboters beim Navigieren einstellen kann. Auch dieser hat einen Wertebereich zwischen 0 und 127 (vgl. Motorsteuerung).
Hält man nun auf der Karte an einer Position den Finger etwas länger gedrückt, so erscheint dort ein zusätzlicher Pin mit der Beschriftung „Ziel“. Die Position dieses Pins in der Weltkarte wird vom Netzwerkthread an den SmartBot-Server in Längen- und Breitengraden übermittelt, welcher diese dann zur Kursberechnung weiter verarbeiten kann.
Betätigt man nun den Startknopf, so wird dies ebenfalls über den Netzwerkthread an den SmartBot-Server übermittelt und der Roboter beginnt zu seinem vorgegebenen Ziel zu fahren. Während der Fahrt kann man nun die Geschwindigkeit variieren, stets die genaue aktuelle Position betrachten und den Webcam-Stream beobachten. Mit einen kurzen Tipp auf „Stop“ hält der Roboter wieder an.

Der Netzwerkthread kommuniziert mit dem Roboter mithilfe simpler Strings. Um die Kommunikation im Programmablauf aufrecht zu erhalten, erfolgt auf jede Nachricht eine Antwortnachricht vom Server, welche entweder „okay“ oder neue Positionsdaten des Roboters beinhaltet, beginnend mit der Zeichenfolge „coord“. Eine typische Server-Client Kommunikationsablauf sieht demzufolge so aus:

		(Socket verbunden)
1.	**Server:** connected;	
2.	**Smartphone:** connected;2000;	(sende alle 2 Sekunden die Position des Roboters)
3.	**Server:** coord;50.733991;7.099813;	(Position des Roboters)
4.	**Smartphone:** okay;	(Bestätigung zur Aufrechterhaltung der Kommunikation) => keine Benutzerinteraktion
5.	**Server:** coord;50.733991;7.099813;	
6.	**Smartphone:** okay;	
7.	**Smartphone:** ziel;50.733992;7.099817;	(Benutzer hat ein Ziel gesetzt)
8.	**Server:** coord;50.733991;7.099813;	
9.	**Smartphone:** okay;	
10.	**Smartphone:** start;	(Benutzer hat Navigation gestartet)
11.	**Server:** coord;50.733991;7.099814;	
12.	**Smartphone:** speed;98;	(Benutzer hat Geschwindigkeit geändert)
13.	**Server:** coord;50.733991;7.099815;	
14.	**Smartphone:** stop;	(Benutzer hat Navigation vorzeitig gestoppt)
15.	**Server:** coord;50.733991;7.099816;	
16.	**Smartphone:** disconnect;	(Benutzer hat App beendet, pausiert, oder Navigation geschlossen) *(Socket geschlossen)*

Abbildung 8: exemplarische Netzwerkkommunikation zw. Smartphone und SmartBot-Server während der selbstständigen Navigation

6.3 Einstellungen

In den Einstellungen konfiguriert man hauptsächlich IP-Adresse und Port des SmartBot-Servers, sowohl für das benutzergesteuerte Fahren, als auch für die Navigation. Andere Werte dienten hauptsächlich Entwicklungszwecken und können so belassen werden wie sie sind.

Abbildung 10: Hauptmenü der App

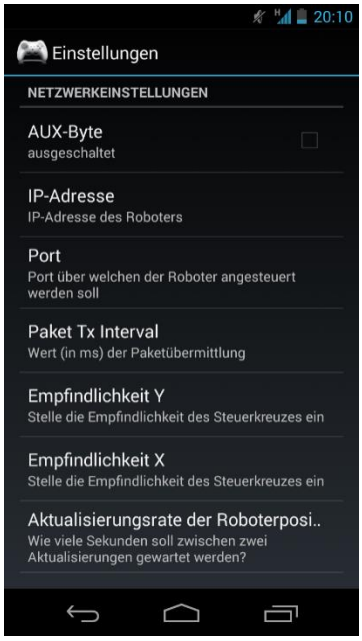


Abbildung 9: Ansicht der App im Menüpunkt "Einstellungen"

7 Beschreibung des SmartBot-Servers

Der SmartBot-Server wird direkt auf dem Raspberry Pi ausgeführt und bildet das Herzstück des Roboters. Neben der oben bereits beschriebenen Verwaltung der einzelnen Gerätekomponenten kümmert er sich auch um die Verarbeitung von Benutzereingaben, welche er über einen UDP-„DatagramSocket“ (manuelle Steuerung) oder einen TCP-„ServerSocket“ (Navigation) von der App entgegennimmt. Beide Schnittstellen haben dann ihre eigenen Routinen zum Umgang mit den Steuerbefehlen.

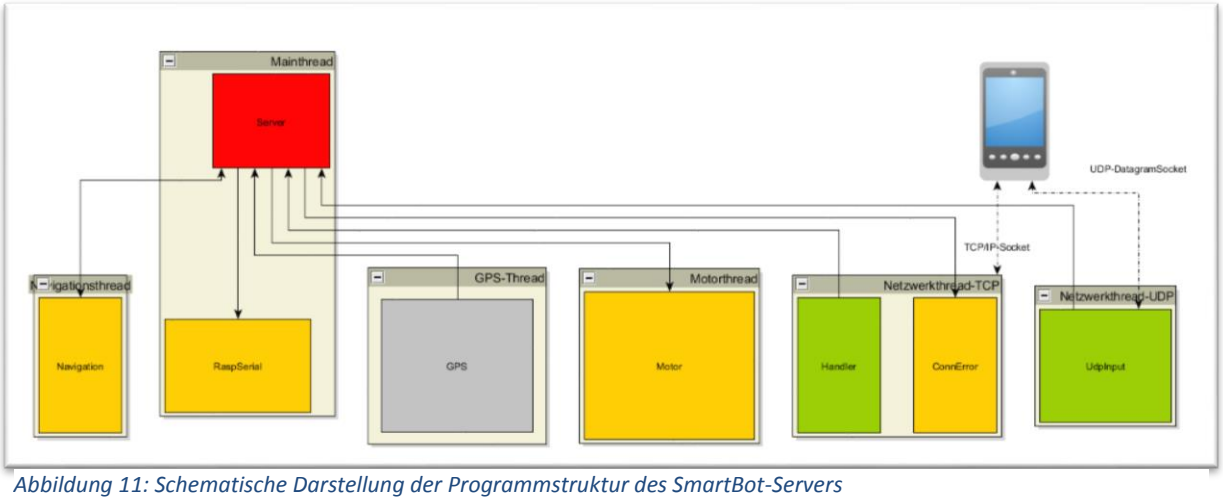


Abbildung 11: Schematische Darstellung der Programmstruktur des SmartBot-Servers

7.1 Manuelle Steuerung

Die eingehenden Befehle sind Positionsangaben der Steuerknüppel im Koordinatensystem, welches seinen Ursprung (0,0) in der Mitte des Knüppels hat. Für eine bessere Handhabung verwenden wir die y-Achse vom linken Steuerknüppel und die x-Achse vom rechten Steuerknüppel. Der linke steuert somit die Geschwindigkeit und der rechte die Lenkung.
Die Routine unterteilt dieses Koordinatensystem in den positiven und negativen Bereich der y-Achse und berechnet dann aus der Kombination von Geschwindigkeit und Richtung den benötigten Geschwindigkeitswert für das einzelne Rad. Der Geschwindigkeitswert wird berechnet, indem von der Grundgeschwindigkeit die gewünschte Richtungsänderung bei einem Rad subtrahiert wird und beim anderen addiert wird.

Beispiel:

$$\textit{Position des Fingers im Koordinatensystem des Joysticks: } \boldsymbol{J\ (70, 20)}$$
$$\textit{das heißt gewünschte Fahrtrichtung: } \boldsymbol{geradeaus\ mit\ Rechtskurve}$$
$$\textit{Wert für linkes Rad: } \mathbf{70\ +\ 20\ =\ 90}$$
$$\textit{Wert für rechtes Rad: } \mathbf{70\ -\ 20\ =\ 50}$$

Aus obiger Rechnung ergibt sich eine Rechtsdrehung.
Für Rückwärtsfahrten müssen Addition und Subtraktion für die Räder getauscht werden.
Die berechneten Werte werden dann an die Motorsteuerung übermittelt und in Radbewegung umgesetzt.

7.2 Selbstständige Navigation

Die Navigationsroutine, also der Teil des Servers, der den Roboter selbständig zu einem Ziel fahren lässt, wurde mithilfe von funktionsorientierter Programmierung gelöst. Bei dieser Art der Programmierung werden die für die Aufgabenstellung nötigen Verhaltensweisen in Funktionen unterteilt und in eine hierarchische Reihenfolge gebracht.

Wir benutzen lediglich die Funktion „fahren“, welche den Roboter einfach geradeaus fahren lässt und die dann bei Bedarf von der Funktion „ausrichten“ abgelöst wird. Diese Funktion richtet den Roboter mithilfe des Kompasses wieder in Zielrichtung aus und beachtet dabei, ob es sinnvoller ist, den eigenen Kurs mit einer Rechts- oder Linksdrehung zu korrigieren. Zuletzt gibt es noch die Funktion „stop“, die die Navigation beendet und den Roboter anhält, wenn das Ziel erreicht wurde, oder sie durch eine Benutzerinteraktion aufgerufen wird. Auf unterster Ebene befindet sich somit die Funktion „fahren“. Sie wird nur aufgerufen, wenn der Roboter in Zielrichtung ausgerichtet ist, das Ziel noch nicht erreicht hat und der Benutzer den Startbefehl gegeben hat. Weicht die Fahrtrichtung vom Ziel ab, so wird, vorausgesetzt der Roboter hat sein Ziel noch nicht erreicht, die Funktion „ausrichten“ aufgerufen und zwar so lange, bis der Roboter wieder in Zielrichtung ausgerichtet ist. Ist der Roboter am Ziel angelangt wird die Funktion „stop“ aufgerufen, die Navigation wird beendet und der Roboter hält an.

8 Bildverzeichnis

Abb. 1: https://commons.wikimedia.org/wiki/File:RaspberryPi.jpg

Abb. 2: https://commons.wikimedia.org/wiki/File:Pulse_wide_wave.svg