RPi-404

Stefan Wyrembek (2065379)

Pieter Haase (2052542)

Volker Thum (2194211)

Dario Schüler (2220045)

Dokumentation

IT-Systeme

Prof. Dr. Edeler

Prof. Dr. Plaß

HAW Hamburg
Fakultät DMI

Inhalt

Konfiguration auf Seiten des Raspberry Pi

Grundkonfiguration Raspberry

Auf dem Raspberry wurde das Betriebssystem Raspbian 4.14 installiert. Für die Ansteuerung der GPIO-Pins wurde die Programmiersprache Python 3 verwendet. Die Kommunikation mit dem Raspberry wird über die Onboard W-Lan Schnittstelle des Pi's realisiert. Der Raspberry wird mittels einer 12V Batterie und einem Spannungswandler (12V zu 5V) betrieben.

Übermittlung von Steuerdaten aus der Android Applikation zu dem Raspberry Pi

Für die Übermittlung von Steuerbefehlen, welche durch die Android Applikation von einem Benutzer erzeugt werden, wird das Nachrichtenprotokoll MQTT (Message Queue Telemetry Transport) verwendet. Dieser Dienst ist aus dem Internet abrufbar. Der MQTT-Server (Broker) empfängt die Steuerbefehle auf einem vorher definierten Kanal, von welchem die Daten von dem Raspberry wieder abgerufen werden können. Für diese Kommunikation ist eine Internetverbindung notwendig, falls auf die Einrichtung eines MQTT Brokers lokal im eigenen Netzwerk oder auf dem Raspberry verzichtet wird. Für die Kommunikation des RPi-404 wurde die Online-Variante gewählt.

Funktion Kamera (camera_turn.py)

Die Kamera ist durch eine 3D-gedruckte Halterung auf die Achse des Stepper Motors montiert. Dieser wird durch den Motortreiber angesteuert, welcher wiederum Steuerbefehle von den GPIO-Pins des Raspberrys erhält. Die Steuerbefehle der GPIO Pins sind abhängig von Steuerbefehlen der Android Applikation, welche anhand eines MQTT Brokers über das Internet gesendet und von dem Raspberry empfangen werden. Die empfangenen Daten werden durch ein Python3 Skript verarbeitet, welches die entsprechenden GPIO Pins ansteuert (Bewegung des Motors links und rechts). Für den Stream des Kamerabildes auf die Applikation des Benutzers wird das für Debian entwickelte Paket „*Motion*" genutzt. Der Service wird auf dem Raspberry gestartet und kann über die IP-Adresse des Pi's mit dem Port „8081" abgerufen werden. Das Konfigurationsskript „*motion.conf*" kann dem Anhang entnommen werden.

Funktion Radar (radar3.py)

Das Radar wurde anhand des „*Ultrasonic*" Ultraschallmoduls realisiert. Der auf einem Stepper Motor montierte Sensor bewegt sich von links nach rechts und misst hierbei anhand von Ultraschallwellen über das Geschwindigkeits-Zeit-Gesetz die Entfernung zu sich in Reichweite befindenden Objekten. Die Berechnung sowie das Schwenken des Moduls wird anhand eines weiteren Python 3 Skripts gesteuert. Die generierten Daten (Winkel und Abstand) werden über MQTT an den Broker gesendet und von der Android Applikation abgerufen und verarbeitet.

Funktion Drive (testfahrt.py)

Die Bewegung des RPi-404 wird anhand von vier Motoren realisiert, welche über eine H-Brücke mit der Batterie verbunden sind. Die H-Brücke empfängt die von dem Python Skript generierten Pulsweiten Modulations Daten, welche über GPIO-Pins gesendet werden und spricht die jeweiligen Motoren an. Das Python 3 Skript verarbeitet Steuerdaten, welche über MQTT von der Android Applikation gesendet werden. Die Steuerdaten bewegen sich in einem Rahmen von [-1 bis 1]. Je nach Wert, nimmt das von dem Skript generierte PWM Signal eine andere Form an und beeinflusst hierdurch die Ausgangsspannung der H-Brücke., was zu einer bestimmten Drehzahl der Motoren führt.

Funktion Kalibrierung

Um sicherzustellen, dass die Kamera und das Radar nach einem Neustart des Raspberry's eine vorher definierte Position einnehmen, (Kalibrierung auf „Null") wird ein Python 3 Methode verwendet,

welches diese Funktion ausführt. Die Halterung der Kamera sowie des Radars sind mit einer Scheibe versehen in der sich jeweils eine Einkerbung befindet. Mittels Motoren bewegen sich die Scheiben durch jeweils eine Gabellichtschranke. Sobald sich die Einkerbung auf der Höhe der Lichtschranke befindet, verändert sich der Widerstand dieser und ein Trigger-Signal wird von einem GPIO Input empfangen. Wenn der GPIO Pin „getriggert“ wird, stoppt der Motor und bewegt sich um einen definierten Winkel in die entgegengesetzte Richtung.

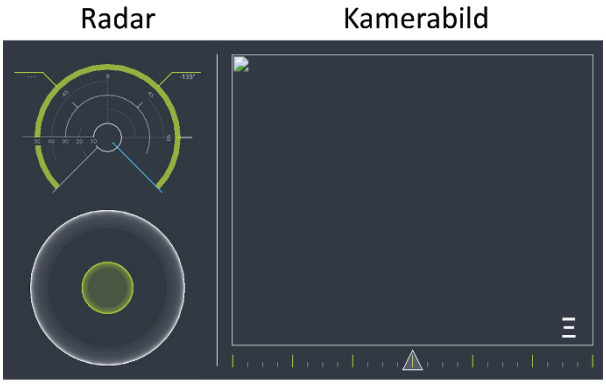
Multrithreading (main.py)

Die oben beschriebenen Skripte werden mittels „*multithreading*“ parallel durch das Skript „main.py“ verarbeitet. Aus den jeweiligen Skripten werden **nur** die Funktionen in die *main.*py importiert, welche benötigt werden. Durch diese Methode wird nur ein Bruchteil der Rechenkapazität des Pi’s benötigt und das System ist weniger störanfällig.

Verwendete Bauteile

Bauteil	**Produktnummer**	**Anzahl**
Stepper Motor	28BYJ-48	2
Treiberplatine für Stepper Motoren	ULN2003	2
Lichtschranke	HY860H-A	2
Motoren	Raspberry DC Motor	4
H-Brücke	L298N	1
Spannungsregler	XL6009	1
Ultrasonic Sensor	HC-SR04	1
Webcam	Speedlink SL-6815-BK	1
Raspberry	Raspberry Pi 3B	1
T-Cobbler	/	1
Roboterplattform	/	1
Gummireifen	/	1
Widerstand	1K Ohm	6
Widerstand	333 Ohm	2
Widerstand	120 Ohm	1
Batterie 12V		1

Android Applikation



Die App besteht im Grunde aus 4 Funktionen, dem *Joystick*, dem *Kamerabild*, der *Kamerasteuerung*, und dem *Radar*. Diese werden von der *MainActivity* der App initialisiert und gesteuert. Im Hintergrund läuft dabei der *MQTTService*, der für das Senden und Empfangen von Steuerbefehlen über einen MQTT Broker zuständig ist. Für die visuelle Gestaltung wurde die API Processing for Android verwendet.

Processing

Processing ist eine API, die es ermöglicht auf recht simple Weise grafische Oberflächen durch Zeichnen von Linien, Kreisen und ähnlichem zu erstellen. Um sie zu verwenden muss sie lediglich in das Projekt eingebunden werden und eine Klasse von *PApplet* erben. Sie implementiert damit automatisch die Methoden *setup()* und *draw()*. In der Setup-Funktion werden Befehle bei der Initialisierung des PApplet-Objekts ausgeführt, die Funktion Draw hingegen wird entsprechend einer definierbaren Refresh-Rate wiederholt ausgeführt.
Es können auch noch Methoden wie beispielsweise *onTouch()* oder *onDrag()* ergänzt werden mit denen Benutzereingaben verarbeitet werden können.
Zur Darstellung innerhalb der App wird aus dem PApplet in der MainActivity ein *PFragment* gemacht, welches einem *View* zugewiesen werden kann.

MQTT Service

Für die Kommunikation der App mit dem Raspberry Pi werden die Steuerwerte der entsprechenden Funktionalität über einen *MQTT-Broker* gesendet. Dafür steht jeder Funktionalität ein eigener Kanal (Topic) zur Verfügung (steering, radar, camera). Die Klasse überwacht die einzelnen Kanäle und gibt die empfangenen Werte an die entsprechende Zielklasse weiter, während mit Hilfe entsprechender Methoden Befehle von den einzelnen Klassen an den Raspberry Pi gesendet werden können.

Joystick

Der Joystick besteht aus dem Joystick selbst (innerer Kreis), welcher bewegt werden kann um den RPi-404 zu steuern und dem möglichen Bewegungsradius (äußerer Kreis) als Hintergrund. Wird der Joystick über eine Touch-Geste aus der Mitte hinausbewegt, werden sowohl der Winkel als auch der Abstand von der Mitte ermittelt und darüber die Steuerbefehle für die Motoren berechnet. Es wird jeweils ein Wert für die Motoren auf jeder Seite des RPi-404 berechnet. Diese bewegen sich im Bereich von -1 (rückwärts) bis 1 (vorwärts). Der Abstand des Joysticks zur Mitte beeinflusst dabei die Geschwindigkeit der Motoren, sodass auch langsames Fahren möglich ist.

Kamerabild

Der Kamera Feed ist das einzige sichtbare Element der App, welches nicht mithilfe der Processing API dargestellt wird. Hierbei handelt es sich um einen einfachen *WebView*, eine Android-eigene Ansicht um Webseiten darzustellen. Als Web Adresse dient hierbei die IP des Raspberry Pi auf dem RPi-404, welcher über die entsprechende Portnummer (8081) das Kamerabild überträgt.

Kamerasteuerung

Um die Kamera an der Front des RPi-404 drehen zu können haben wir zwei verschiedene Möglichkeiten in der App implementiert. Zum einen kann die Kamera über einen *Slider* auf dem Bildschirm mit einer Touch-Geste um den entsprechenden Winkel und die entsprechende Richtung gedreht werden. Des Weiteren kann die Kamera durch das Berühren des Kamerabilds und gleichzeitiger Drehung des Smartphones um dessen *Pitch-Achse* (vom linken Bildschirmrand zum Rechten) bewegt werden. Da die Lagesensoren ihre Messwerte teilweise recht sprunghaft ändern, wurden diese durch einen Lowpass gefiltert, welcher große Sprünge zwischen Messwerten effektiv vermindert und so eine gleichmäßigere Steuerung der Kamera ermöglicht.
Zu Beginn des Projekts haben wir zur Steuerung ein Smartphone, welches über einen Gyroskop-Sensor verfügt, verwendet. Dies ist leider im Laufe des Projekts ausgefallen, woraufhin wir zur Steuerung ein Tablet verwendet haben, welches leider nicht über ein Gyroskop verfügte. Der Code der App wurde dahingehend angepasst, dass zur Berechnung der Lage des Tablets dessen Beschleunigungs- und Geomagnetischer Sensor (Kompass) benutzt wird. Leider hat sich gezeigt, dass diese Art der Lageberechnung zwar recht gut bei einem liegenden Gerät funktioniert, für ein in der Hand gehaltenes Gerät im *Landscape-Modus* allerdings nur unzureichende Ergebnisse liefert.

Radar

Die Messwerte des Ultraschall-Sensors werden im Stile eines Radars, wie es beispielsweise auf Schiffen oder in der Flugüberwachung verwendet wird, visualisiert. Der Ultraschallsensor überträgt den gemessenen Abstand in Zentimetern für den aktuellen Abtastwinkel über MQTT. Die App speichert den Distanz-Messwert zu jedem Winkel in einem Array von Float-Werten. Objekte mit einem Abstand von weniger als 50 Zentimeter werden durch eine Linie auf dem Radarbildschirm dargestellt. Zusätzlich dient ein Kreis, dessen Radius dem aktuell gemessenen Abstand entspricht, dazu anhand der Skala auf der linken Hälfte des Radars den genauen Abstand besser ablesen zu können. Zur Visualisierung der aktuellen Messrichtung des Ultraschallsensors läuft ein „Zeiger“ mit dem Messwinkel mit.
Um die Performance zu optimieren wird der „Hintergrund“ des Radars lediglich bei der Initialisierung des Objekts gezeichnet und in einen *Image-Buffer* geladen, dessen Inhalt bei jedem Aufruf der Draw-Funktion ausgelesen und als Hintergrund gezeichnet wird.

Tauschen von Radar und Kamerabild mittels Swipe-Geste

Für eine größere Darstellung des Radars können die Positionen des Kamerabilds und des Radars mit einer Swipe-Geste auf dem Radarbild nach unten-rechts (in Richtung der Mitte des Kamerabilds) oder auf dem Kamerabild nach oben-links getauscht werden. Dadurch wird eine Ansicht jeweils vergrößert auf der rechten Seite des Bildschirms angezeigt und die andere Ansicht entsprechend verkleinert auf der linken Seite.
Da jedes *PApplet* einen Eigenen View verwendet und diese nicht ohne erheblichen Programmieraufwand zur Laufzeit getauscht werden können, wurde diese Funktionalität durch einen kleinen Trick realisiert. Das Radar sowie das Kamerabild existieren in beiden Views zweimal und es wird lediglich deren Sichtbarkeit ein- und ausgeschaltet. Um Ressourcen zu schonen wird das Zeichnen des Radars, welches nicht sichtbar ist, pausiert.
Es ergab sich dadurch der Vorteil, dass den beiden Instanzen des Radars im Konstruktor direkt übergeben werden konnte, ob es sich um die normale (*„SMALL“*) oder die vergrößerte (*„LARGE“*) Version handelt und so Linienbreiten und Schriftgrößen entsprechend angepasst werden konnten.