Der RP6-Robot mit einem ESP als WiFI-Remote

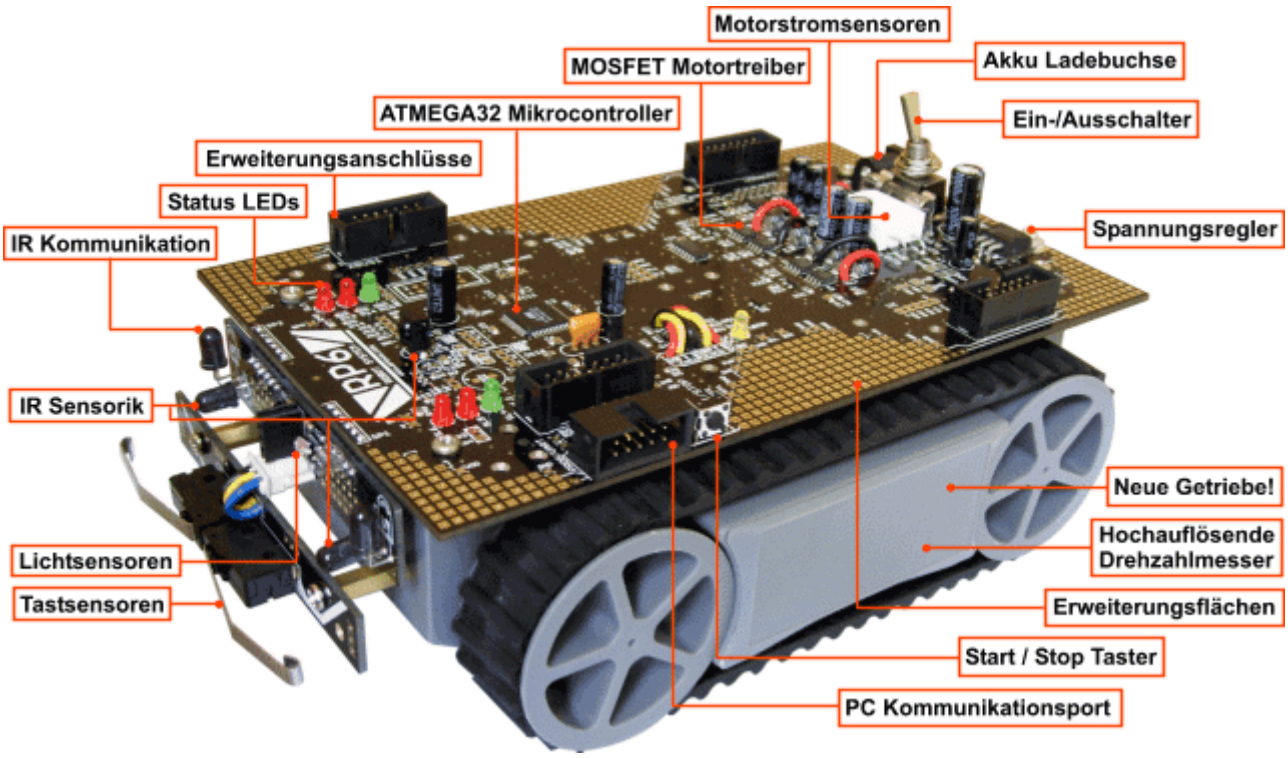
Der RP6 ist (war) eine spannende Roboter-Plattform der Firma Arexx. Leider wird er nicht mehr weiterentwickelt und auch ein Nachfolger bleibt wohl aus.
Doch mit einem ESP32 lässt sich der (alte, aber gute) RP6 wieder auf den Stand der Technik bringen!

- Der RP6 - in Hard- und Software
- Remotrol von FabianE.
- Der neue RP6-ESP
- Lehrkonzept

Der RP6 - in Hard- und Software

Der [RP6 von Arexx] (https://www.arexx.com/rp6/html/de/) ist mit allerlei Komponenten und einem sehr schönen Stück Software ausgestattet.

Der RP6 bietet



- Eine Plattform mit Kettenantrieb
- Zwei starke Motoren mit Getriebe
- je Motor ein Drehzahlsensor und ein Stromsensor (dadurch kann etwa auch ein eingeklemmter Finger "gespürt" werden)
- zwei Helligkeitssensoren vorne (um etwa einer Lichtquelle zu folgen)
- zwei Bumper vorne
- IR-Abstandssensorik vorne
- IR-Kommunikation
- sechs LEDs mit vier freien IOs
- zwei freie ADCs

Erweiterungen

Es gibt für den RP6 diverse Erweiterungen. Für diese Erweiterungen bietet der RP6 vorne und hinten jeweils einen vordefinierten Erweiterungsport (genannt "XBUS") und einen vom User frei belegbaren Erweiterungsport (genannt "USRBUS").

- Eine Prototypenplatine
- M32-Control
- M128-C-Control
- M256-Wifi-Control Diese Erweiterungen beinhalten in erster Linie je einen stärkeren Prozessor, mehr IOs und ADCs, Taster, LEDs, Speicher, Buzzer, Sensoren, ... oder im Falle der M256 ein WiFi-Modul.

nicht-kommerzielle Erweiterungen

fabqu hat vor einigen Jahren mit Dirk aus dem Roboternetz zwei Erweiterungen entwickelt:

- MultiIO
- ArduIO Sie bieten viele weitere IOs, ADCs, DACs, PWMs, Hochleistungstreiber, Akku-Versorgungen, Sensorik (Uhr, GPS, IMU, Temperatur, Strom, Spannung, Licht, Abstandssensorik, ...) und die ArduIO dient der Kombination von Arduino-Boards und -Modulen mit dem RP6. Außerdem gab es weitere Boards wie Lichter, Blinklichter, Stoßstangen, Liniensucher-Module, Radio, ...

Software/Firmware

Die Software und Firmware (mit allen RP6-Bibliotheken und auch Manuals) gibt es HIER bei Arexx herunterzuladen.

Das Programmers Notepad ist leider recht veraltet, tut aber noch seinen Dienst. Im Manual des RP6 ist beschrieben, wie man es aufsetzt und wie man die Firmware kompiliert. Anders als im Arduino-Universum geht hier noch alles "von Hand", man benötigt also ein echtes MakeFile. Arduino übernimmt all diese etwas komplizierten Dinge. Auch die Firmware selbst ist in echtem C geschrieben, nicht im vereinfachten Arduino-C.

Die Formware ist richtig gut gemacht. Hier gibt es sehr umfangreiche Bibliotheken, vor allem für den RP6-Base. Etwa kann den Motoren eine Geschwindigkeit gegeben werden und der Roboter hält

diese automatisch, gleicht rechts/links ab und erkennt etwa auch einen eingeklemmten Finger! Beim Starten/Stoppen/Umkehren werden die Motoren nicht hart geschaltet, sondern etwas weicher gebremst/beschleunigt. Das schont das Getriebe!

Kompatibilitätsprobleme

`Achtung:` der RobotLoader, auch in seiner aktuellsten Version, lässt sich zunächst unter neueren Windows-Systemen nicht installieren. Hierfür bitte NICHT wie empfohlen die 64bit-Variante von Java herunterladen, sondern die 32bit-Variante. Nur damit klappt der RobotLoader.

Remotrol von FabianE.

Im Roboternetz hatte FabianE. eine recht umfangreiche Fernsteuerung präsentiert, welche in Version 1.3 recht gut lauffähig war und sehr viel abdeckte und auch mit allen Erweiterungen kombiniert werden konnte. Den damaligen Forumseintrag gibts HIER.

fabqu hat die Version 1.3 bei sich und wird diese, falls FabianE. es erlaubt, erneut in etwas ageänderter Variante veröffentlichen.

Die Firmware auf dem RP6 war als "Slave" gedacht und erwartete im Wesentlichen über Bluetooth Befehle vom PC und schickte Sensordaten darüber an den PC. Dabei kommuniziert der RP6 über seine UART-Schnittstelle (universal asynchronous receive/transmit), also Rx (receive data) und Tx (transmit data). Je nachdem, welche Plattform (RP6-Base, M32, M128 oder M256) man verwendet, werden entsprechend alle verfügbaren Sensordaten routinemäßig über die UART-Schnittstelle gesendet. Die `Baudrate ist dabei 38.400`. Die Befehlsstruktur, welche der RP6 erwartet, ist etwas komplizierter.

Senden von Befehlen an den RP6

Die Kommandos sind vom Typ `#E1:(V1):(V2):E2:id*` -> Beginn ist `#`, Ende ist `*`. `id` ist ein Counter, der stets inkrementiert werden muss und maximal 99 sein darf. `E1` und `id` sind erforderlich. `E2` ist obligatorisch. `V1` und `V2` sind weitere Parameter, die es nicht immer braucht.

Erste Ebene	E1	Zweite Ebene	E2
#define CMD_SET_SPEED	1	Features:	
#define CMD_SET_SERVO	2	#define SET_FEATURE_GENERAL	0
#define CMD_SET_LEDS	3	#define SET_FEATURE_LIGHT	1
#define CMD_SET_BEEP	4	#define SET_FEATURE_INT0	3
#define CMD_SET_START_MELODY	5	#define SET_FEATURE_MIC	4
#define CMD_SET_FEATURE	6	#define SET_FEATURE_SRF08_RADAR	5

Erste Ebene	E1	Zweite Ebene	E2
#define CMD_SET_STOP	7	#define SET_FEATURE_SRF02	6
#define CMD_SET_CONNECTION_SPEED	8		
#define CMD_GET_FIRMWARE	9	LEDs:	
#define CMD_SET_ACSPOWER	10	#define LEDS_RP6	0
#define CMD_SET_TEST	11	#define LEDS_M32	1
#define CMD_RESET_ID_COUNTER	99		
x		ACS:	
x		#define ACS_POWER_OFF	0
x		#define ACS_POWER_LOW	1
x		#define ACS_POWER_MED	2
x		#define ACS_POWER_HIGH	3
x			
x		Tests	
x		#define TEST_LCD	0
x		#define TEST_BEEPER	1
x		#define TEST_LED	3
x		#define TEST_EXTERNAL_MEMORY	4
x		#define TEST_I2CLED	5
x		#define TEST_I2CMOTOR	6
x		#define TEST_MIC	7
x		#define TEST_MOTOR	8
x		#define TEST_BATTERY	9
x		#define TEST_ACS	10
x		#define TEST_BUMPER	11
x		#define TEST_LIGHTSENSOR	12

Beispiele:

- `#7:21*` -> Stop, mit id=21
- `#1:50:75:0:22*` -> fahren (E1=1) vorwärts (E2=0) mit Geschwindigkeit V1=50 linke Kette, V2=75 rechte Kette (von 200) und id=22
- `#3:0:33:23*` -> LED (E1=3) auf der RP6-Base (E2=0), es warden die Status LEDs SL1 (Bit 1=1=1) und SL6 (Bit 6=1=32) angeschaltet (V1=1+32=33)

Empfangen von Daten vom RP6

Die Daten des RP6 kommen in Blöcken zu mehr oder weniger festen Zeitintervallen. Die Blöcke sehen wie folgt aus:

```
[DATA]
Bat:955
SpeedL:0
SpeedR:0
PowerL:0
PowerR:2
LightL:592
LightR:865
BumpL:0
BumpR:0
ObsL:1
ObsR:1
ADC0:860
ADC1:855
BumpHL:0
BumpHR:0
[/DATA]
```

Sie beginnen also mit `[DATA]` und enden mit `[/DATA]`, dazwischen stehen die Werte in je einer Zeile in Tupeln aus einem Keyword und dem Wert; beide sind durch Doppelpunkt `:` getrennt. Beim Senden wird kein Interrupt o.ä. ausgelöst, man muss also auf der anderen Seite stets horchen und den Puffer bei Bedarf lesen.

Der neue RP6-ESP

Hier soll der RP6 (genauer der RP6v2-Base) mit einem ESP kombiniert werden.

Hardware

- RP6v2
- Bumper-Board der MultiIO, mit zwei Bumpern und zwei Sharp-IR-Abstandssensoren sowie zugehörigem Transistor
- LED-Platinen der ArduiIO
- auf der Basis werden zwei BC847 aufgebaut, um die Scheinwerfer über die Status-LEDs der RP6-Base zu schalten; die Blinker werden direkt an die Status-LEDs der RP6-Base angeschlossen. Scheinwerfer Front/Heck innen (lila Kabel, IO2), Scheinwerfer Front/Heck außen (grünes Kabel, IO5), Blinker links (oranges Kabel, IO4) und Blinker rechts (gelbes Kabel, IO1).
- Die Bumper (ON-L und ON-R) sollen noch an die IO1 (rechts) und IO5 (links) angeschlossen werden. Außerdem der Transistor vor den Sharp-Abstandssensoren auf LIO1 (dort ist auch der rechte vordere Bumper angeschlossen). Und die beiden Sharp-Abstandssensoren kommen auf ADC0 und ADC1.
- ESP32 mit 4fach-Pegelwandler auf Erweiterung. Der ESP kommuniziert via UART (Rx2/Tx2) mit dem RP6. Die Baudrate beträgt 38.400.
- Der ESP ist wie folgt angeschlossen: Pin Rx2 (GPIO16) via Pegelwandler auf Tx des RP6; Pin Rx2 (GPIO17) via Pegelwandler auf Tx des RP6. Pin D21 auf ST2 des RP6. Pin D19 via Pegelwandler auf ST1 des RP6.
- ST1/ST2 des RP6 starten/resetten den RP6: zieht man ST1 auf +5V (HIGH) startet der RP6. Zieht man den ST2 auf GND (LOW) so resettet er.

Firmware RP6

Wie beschrieben kommt im Wesentlichen die Firmware für den RP6-Base von FabianE.'s Remotrol zum Einsatz. Hinzugefügt wurde in der RP6RobotBaseLib.c der Absatz zu den BackBumpers:

```
//*********NEW***********
// from: nil.at, RP6-Forum


uint8_t getBackBumperLeft(void)
```

```
{
  PORTB &= ~SL4;□□□□//Schalte StatusLED4 (liegt auf PortB) ab
  DDRB &= ~SL4; □□□□//Schalte StatusLED4 auf Eingang
  nop(); □□□□□□//warte kurz
  uint8_t tmp = PINB & SL4; □//Lege den Wert von StatusLED4 auf "tmp"
  if(statusLEDs.LED4) □□□//wenn StatusLED4 vorher schon an war
  {
    DDRB |= SL4; □□□□//Schalte StatusLED4 auf Ausgang
    PORTB |= SL4; □□□//und schalte StatusLED4 wieder an
  }
  return tmp;
}

uint8_t getBackBumperRight(void)
{
  PORTC &= ~SL1; □□□□//Schalte StatusLED1 (liegt auf PortC) ab
  DDRC &= ~SL1; □□□□//Schalte StatusLED1 auf Eingang
  nop(); □□□□□□//warte kurz
  uint8_t tmp = PINC & SL1; □//Lege den Wert von StatusLED1 auf "tmp"
  if(statusLEDs.LED1) □□□//wenn StatusLED1 vorher schon an war
  {
    DDRC |= SL1; □□□□//Schalte StatusLED1 auf Ausgang
    PORTC |= SL1; □□□//und schalte StatusLED1 wieder an
  }
  return tmp;
}
```

Entsprechend wurde auch der `BUMPERS_stateChanged_DUMMY(void)` angepasst und die RP6RobotBaseLib.h.

Software ESP32

Der auf dem ESP32 laufende Code findet sich am Ende dieses Artikels. Die Software des ESP32 wurde zunächst mit RemoteXY gemacht. RemoteXY liefert ein schönes GUI für das Smartphone, jedoch mit allerlei Einschränkungen (nur gewisse Bausteine; keine Einbindung eines Kamerabildes; bei mehr als fünf Bausteinen wird es mit 12,99 Euro kostenpflichtig!). Daher wurde doch wieder auf eigenen Code zurückgegriffen. Nun dient ein M5stickC mit JoyC-Joysticks als Remote-Controller sowie zur Anzeige der Sensordaten. Auf dem ESP32 des RP6 läuft ebenfalls eigener Code, welcher im Wesentlichen die Sensordaten via ESPnow an den M5stickC schickt und vom M5stickC Befehle

erwartet. Der Code wurde von Fabian geschrieben.

Library für den RP6 mit UART2

Es gibt eine Bibliothek RP6_UART2_LIB.h, welche es dem ESP32 ermöglicht, via UART2 mit dem RP6 zu kommunizieren.

Änderungen und Erweiterungen

Einiges mehr soll noch kommen, hier ein paar Ideen/Pläne:

Hardware

- ☐ der ESP32 soll auf eine richtige Erweiterungsplatine, mit Pegelwandler, I2C- und UART-Zugang und USRBUS/XBUS
- ☐ Linienfolgermodul der RP6-MultiIO mit Servo, um es hoch- und herunterzufahren
- ☐ Akku mit Ladeeinheit
- ☐ Ultraschall-Abstandssensor mit Drehturm und Servo
- ☐ evtl. ESP32-CAM mit in den Drehturm
- ☐ evtl. Scheinwerfer mit in den Drehturm
- ☐ Gehäuse
- ☐ OLED-Display
- ☐ IMU-BreakoutBoard

Firmware RP6

Weitere Änderungen sollen sein: Wenn das ACS (Anti-Collision-System) an ist, soll automatisch beim Rückwärtsfahren das vordere ACS de- und das hintere (die beiden IR-Sharp-Sensoren) aktiviert werden; beim Vorwärtsfahren entsprechend das hintere de- und das vordere aktiviert. Bei einer Drehung auf der Stelle sollen beide aktiv sein.

Außerdem soll bei Kurven der jeweilige Blinker blinken und bei Drehungen auf der Stelle sowie bei Zusammenstößen soll der Warnblinker an gehen.

Software ESP32

- ☑ Reset des RP6 (dadurch Stop) nach Verbindungsabbruch
- ☑ Anzeige diverser Sensoren auf der Fernsteuerung (M5stickC)

- ☐ Einbindung des Kamerabildes?
- ☐ automatisches ACS, wodurch etwa bei Gefahr automatisch angehalten wird
- ☐ Routinen (Lichtfolger, Abstandhalter, Labyrinth, evtl. auch mit AI-Komponenten wie HIER)

Code auf dem ESP32

diesen gibt's demnächst auf der Github-Seite als Beispielcode zur UART-Library RP6_UART2_LIB.h.

Lehrkonzept

Die fertige Software und Firmware könnte man auf die anderen vorhandenen RP6 birngen und je mit einem ESP32 kombinieren. Somit könnten in ESP32-Kursen auf Arduino-Niveau und mit evtl. RemoteXy schöne Projekte umgesetzt werden!