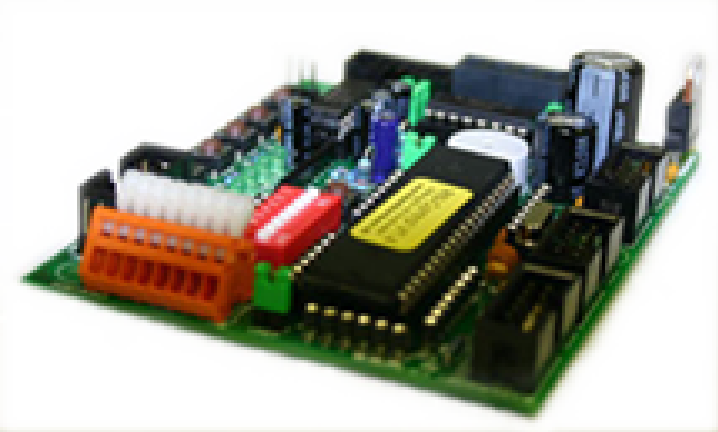
RN-Control ARDUINO 1.x Bibliothek
ATMEL 32

Version 1.1



Bernd Klein

14. Oktober 2012

Inhaltsverzeichnis

Kapitel 1

ARDUINO 1.x - RN-Control Library

1.1 Bibliothek RNControl

Um das RN-Control 1.4 basierend auf einen ATMEL 32 auch mit der ARDUINO-Software nutzen zu können, bedarf es der Anpassung diverser ARDUINO Core-Dateien (siehe auch **??**)sowie die Einbindung der RNControl-Bibliothek. Dieses Dokument beschreibt die Nutzung der RNControl-Bibliothek.

Zusätzlich wurde der von von uns verwendete ISP-Programmer als eigener Programmer in die Datei „programmers.txt aufgenommen.
Diese Dokumentenversion basiert auf die Arduino IDE 1.0.1

1.2 Einführung

Die ARDUINO-Software und die ARDUINO-Boards stellen eine Reihe von Befehlen zur Verfügung, die das Arbeiten mit einem Mikrocontroller sehr vereinfachen. Zusätzlich sind eine Vielzahl an Bibliotheken verfügbar, die das Arbeiten mit externen Geräten (Displays, Ethernet, I2C, OneWire, . . .) unterstützen.

Alle Befehle, die auf das Schreiben/Lesen auf PORTS sich beziehen haben die Gemeinsamkeit, dass die Ports nicht nach den Portnamen (PORTA/PORTB/PORTC/PORTD/...) aufgebaut sind, sondern, dass die verfügbaren Pins einfach durchnummeriert sind. Die Erweiterung der ARDUINO-Software um den ATMeaga 32 nutzt die gleiche Funktionalität. Das hat zur Konsequenz, dass die Ports auf dem RN-Control mit den Arudino-Befehlen angesprochen werden können. Eine genau Beschreibung der Zuordnungen ist im nächsten Kapitel beschrieben.

WICHTIG

Um mit der RNControl-Bibliothek arbeiten zu können, muss diese in den Sketch eingebunden werden. Am Anfang des Sketches

```
#include <RNControl.h>
RNControl rn;
```

einfügen.
Im Sketch kann die Bibliothek z.b. mit ***rn.stopMotor();*** genutzt werden.

Kapitel 2

Installation

Die Installation der RN-Control Bibliothek ist relativ einfach. Leider unterstützt die derzeitige Arduino-Software nicht den ATMEGA 32 Microcontroller. Somit müssen mehrere Dinge durchgeführt werden

1. Kopieren der RNControl Library in den „libraries“ in
   arduino-1.0.1/libraries/RNControl
   Der Ordner RNControl muss angelegt werden

2. Kopieren der Datei *pins_arduino.h* Datei in
   arduino-1.0.1/hardware/arduino/variants/mega32
   Der Ordner *mega32* muss angelegt werden
   Info: Diese Datei enthält die notwendigen Port-Zuordnungen

3. Kopieren der Datei *boards.txt* in
   arduino-1.0.1/hardware/arduino/
   Info: Diese Datei enthält die Zuordnung das es innerhalb der IDE das Board RNControl verfügbar ist

4. Kopieren der Datei *programmers.txt*
   arduino-1.0.1/hardware/arduino/
   Info: Diese Datei enthält einen zusätzlichen USB-ISP Programmer (AVR stk500v2)

Kapitel 3

Portzuordnungen

Die Arduinosoftware stellt diverse Befehle zur Verfügung, die eine Manipulation der Ports bzw. der Pins durchführen. Auf den original Arduinoboards sind diese Pins durchnummeriert. Diese Nummerierung kann auch für das das RN-Control angewendet werden.

Arduinobefehle:

1. *void pinMode(id, INPUT/OUTPUT)*
   Konfiguriert den Port als Ein- oder Ausgang. Muss gesetzt werden, bevor einer der nächsten Befehle aufgerufen wird
2. *byte digitalRead(id)*
   Liest vom Digitalport *id* den aktuellen Status ein. Liefert HIGH/LOW (1/0) zurück.
3. *void digitalWrite(id, status)* Schreibt auf den Digitalport *id* einen neuen *status*. Am einfachsten können hier die Konstanten HIGH(1) oder LOW(0) verwendet werden.
4. *int analogRead(id)*
   Liest den Analogport *id* ein und liefert einen Wert zwischen 0-1023 zurück. Alle Analogports des RN-Controls liefern einen 10Bit Analogwert zurück.
5. *void analogWrite(id, value)* Setzt den Analogpin *id* auf den Wert *value*. *Value* kann einen Wert zwischen 0 und 1023 sein. Analogwerte werden über die PWM-Funktionalität des ATMega-Controllers erzeugt.

3.1 Port Übersicht

3.2 PORT A - Analogport (ID 24-31)

Zum Ansprechen des Port-A und dessen acht Pins muss nachfolgende Nummerierung angewendet werden:
Port A ist der Analogport des RN-Control Boards. Er kann auch als Digitalport verwendet werden, falls keine analogen Signale verarbeitet werden müssen. Ausnahme: PortA Pin 7, dieser Pin ist hart verdrahtet mit den Tastern auf dem Board.

Port	HWPin	ArduinoPin	Bedeutung
PORT A	40	D30	A0, Analog Input/Output
PORT A	39	D31	A1, Analog Input/Output
PORT A	38	D24	A2, Analog Input/Output
PORT A	37	D25	A3, Analog Input/Output
PORT A	36	D26	A4, Analog Input/Output
PORT A	35	D27	A5, Analog Input/Output
PORT A	34	D28	A6, Analog Input/Output
PORT A	33	D29	A7, Abfrage der Tasten auf dem RN-Control

Tabelle 3.1: Port A

3.3 PORT B - Digitalport (ID 8-15)

Zum Ansprechen des Port-B und dessen acht Pins muss nachfolgende Nummerierung angewendet werden:

Port	HWPin	ArduinoPin	Bedeutung
PORT B	1	D8	PB0, Digital Input/Output, Vorbelegung: Motor 2, Kanal 1
PORT B	2	D9	PB1, Digital Input/Output Vorbelegung: Motor 2, Kanal 2
PORT B	3	D10	PB2, Digital Input/Output Vorbelegung: INT2, wird auch von I2C als Alarmleitung genutzt
PORT B	4	D11	PB3, Digital Input/Output OC, steht zur freien Verfügung
PORT B	5	D12	PB4, Digital Input/Output SS, steht zur freien Verfügung
PORT B	6	D13	PB5, Digital Input/Output Vorbelegung: MOSI, wird von ISP genutzt
PORT B	7	D14	PB6, Digital Input/Output Vorbelegung: MISO, wird von ISP genutzt
PORT B	8	D15	PB7, Digital Input/Output Vorbelegung: SCK, wird von ISP genutzt

Tabelle 3.2: Port B

3.4 PORT C- Digitalport (ID 16-23)

Zum Ansprechen des Port-C und dessen acht Pins muss nachfolgende Nummerierung angewendet werden:

Port	HWPin	ArduinoPin	Bedeutung
PORT C	22	D16	PC0, Digital Input/Output, Vorbelegung: SCL, I2C
PORT C	23	D17	PC1, Digital Input/Output Vorbelegung: SDA, I2C
PORT C	24	D18	PC2, Digital Input/Output TCK, steht zur freien Verfügung
PORT C	25	D19	PC3, Digital Input/Output TMS, steht zur freien Verfügung
PORT C	26	D20	PC4, Digital Input/Output TDO, steht zur freien Verfügung
PORT C	27	D21	PC5, Digital Input/Output TDI, steht zur freien Verfügung
PORT C	28	D22	PC6, Digital Input/Output Vorbelegung: wird von Motor 1 Kanal 1 genutzt
PORT C	29	D23	PC7, Digital Input/Output Vorbelegung: wird von Motor 1 Kanal 2 genutzt

Tabelle 3.3: Port C

3.5 PORT D- Digitalport (ID 0-7)

Zum Ansprechen des Port-D und dessen acht Pins muss nachfolgende Nummerierung angewendet werden:

Port	HWPin	ArduinoPin	Bedeutung
PORT D	14	D0	PD0, Digital Input/Output, Vorbelegung: RXD, RS232
PORT D	15	D1	PD1, Digital Input/Output Vorbelegung: TXD, RS232
PORT D	16	D2	PD2, Digital Input/Output INT0, steht zur freien Verfügung
PORT D	17	D3	PPD3, Digital Input/Output INT1, steht zur freien Verfügung
PORT D	18	D4	PD4, Digital Input/Output Vorbelegung: OC1B, PWM Motor 1
PORT D	19	D5	PD5, Digital Input/Output Vorbelegung: OC1A, PWM Motor 2
PORT D	20	D6	PD6, Digital Input/Output ICP, steht zur freien Verfügung
PORT D	21	D7	PD7, Digital Input/Output OC2, steht zur freien Verfügung

Tabelle 3.4: Port D

Kapitel 4

Motoren

Um die Nutzung der Motoren deutlich zu vereinfachen stellt die RNControl-Bibliothek drei Methoden zur Verfügung. Zusätzlich stehen eine Reihe von Konstanten für das Arbeiten mit den Motor-Befehlen zur Verfügung (siehe Kapitel 5).

4.1 Methode *setMotorOffset()*

Mit dieser Methode kann man pro Motor einen negativen oder positiven Offset auf die gewählte Geschwindigkeit aufschlagen. Dies kann dazu verwendet werden, wenn zwei Motoren bei gleichem Speed aber unterschiedlich schnell drehen und somit der Roboter eine leichte Kurve fährt. **Der angegebene Wert wird als prozentualer Anteil der gewählten Geschwindigkeit verwendet.**

Der Offset kann im Bereich von *MIN_OFFSET*/*MAX_OFFSET* liegen.

Default: 100

Befehl:

```
void setMotorOffset(uint8_t id, uint8_t offset)
```

Beispiel:
(Effektiv *MOTOR_ONE* mit einer Geschwindigkeit von 165 (=10 von 150) ansteuern)

```
{
   setMotorOffset(MOTOR_ONE, 10);
   fwMotor(MOTOR_BOTH,150); // effektiv 165 => 150 + 10%

   setMotorOffset(MOTOR_ONE, -10);
```

```
   fwMotor(MOTOR_BOTH,150); // effektiv 135 => 150 - 10%
}
```

4.2 Methode *setMotorReverse()*

Beim Bau des Roboters und beim Anschluß der Motoren, kommt es häufig vor, dass die Anschlußkabel vertauscht werden, dann läuft der Motor plötzlich ein eine andere Richtig als gedacht. Um nun es zu vereinfachen, kann der Befehl *setMotorRevers* eingesetzt werden, dieser Befehl tauscht die Drehrichtung des ausgewählten Motors.

Wird für den Parameter *reverse* als Value *true* verwendet, wird die Drehrichtung des Motors geändert. Mit *false* wird wieder die ursprüngliche Drehrichtung ausgewählt.

Default: false

Befehl:

```
void setMotorReverse(uint8_t id, bool reverse)
```

Beispiel:
(Effektiv *MOTOR_ONE* soll die Drehrichtung ändern)

```
{

   fwMotor(MOTOR_ONE,127);            // Motor dreht z.B. CW

   setMotorReverse(MOTOR_ONE, true);
   fwMotor(MOTOR_ONE,127);            // Motor dreht jetzt CCW

   setMotorReverse(MOTOR_ONE, false);
   fwMotor(MOTOR_ONE,127);            // Motor dreht jetzt CW

}
```

4.3 Methode *motor()*

Generische Methode um einen Motor *id* in der Geschwindigkeit *speed (0-255)* mit der Richtung *dir* drehen zu lassen.

Befehl:

```
void motor(uint8_t id, uint8_t speed, uint8_t dir)
```

Anwendung: man möchte die komplette Kontrolle über die Motorsteuerung nutzen. Diese generische Methode wird von den beiden nächsten Methoden genutzt.

Beispiel:
(linker Motor mit Geschwindigkeit 255 vorwärts drehen)

```
{
   motor(MOTOR_LEFT,255,MOTOR_DIR_FORWARD);
}
```

4.4 Methode *motorExt()*

Generische Methode mit der beide Motoren angesteuert werden können. Die Parameterwerte können von -255 bis +255 angegeben werden. Negative Werte lassen den Motor entgegengesetzt drehen.

Befehl:

```
void motorExt(int16_t id, int16_t speed)
```

Anwendung: Je nach Verwendungszweck kann es einfacher sein mit einem Befehl beide Motoren getrennt anzusteuern und über den Geschwindigkeitswert zusätzlich die Drehrichtung anzugeben.

Beispiel:
(MOTOR_ONE mit Geschwindigkeit 255 CW drehen, MOTOR_TWO mit 255 CCW)

```
{
   motorExt(255,-255);
}
```

4.4.1 Methode fwMotor()

Methode um den Motor *id* in der Geschwindigkeit *speed* vorwärts drehen zu lassen

Befehl:

```
void fwMotor(uint8\_t id, uint8\_t speed)
```

Beispiel:
(linker Motor vorwärts mit Speed 127)

```
{
   fwMotor(MOTOR_LEFT,127);
}
```

4.5 Methode bwMotor()

Methode um den Motor *id* in der Geschwindigkeit *speed* rückwärts drehen zu lassen

Befehl:

```
void bwMotor(uint8_t id, uint8_t speed)
```

Beispiel:
(linker Motor rückwärts Speed 127, rechter Motor rückwärts Speed 200)

```
{
   bwMotor(MOTOR_LEFT,127);
   bwMotor(MOTOR_RIGHT,200);
}
```

4.6 Methode arcMotors()

Läßt dem Roboter im Kreisbogen fahren. Entweder im Uhrzeigersinn oder entgegen dem Uhrzeigersinn.
Die Parameter:
arcDIR CW/CCW (siehe Konstanten)
arcRadius Wert zwischen 0 - *MAX_ARC_RADIUS* (0 = drehen auf der Stelle). Je höher der Wert je größer der Radius
speed Geschwindigkeit für den Kreisbogen

Befehl:

```
void arcMotors(uint8_t arcDir, uint8_t arcRadius, uint8_t speed)
```

Beispiel:
(Roboter im Uhrzeigersinn einen Kreisbogen fahren (mittlerer Radius) mit Speed 127)

```
{
   arcMotors(MOTOR_ARC_CW, 5, 127); // Kreisbogen
   arcMotors(MOTOR_ARC_CW, 0, 127); // drehen auf der Stelle
}
```

4.7 Methode stopMotor()

Methode um den Motor *id* zu stoppen

Befehl:

```
{
   void stopMotor(uint8_t id)
}
```

Beispiel:

```
{
   stopMotor(MOTOR_BOTH); // beide Motoren stop
}
```

4.8 Verwendung der Konstante MOTOR_BOTH

Um beide Motoren gleichzeitig anzusprechen, kann auch die Motorkonstante *MOTOR_BOTH* verwendet werden. Die Konstante kann bei allen Motorbefehlen als Wert für den Parameter *id* verwendet werden.

Beispiel: (linker und rechter Motor vorwärts Speed 127)

```
{
   fwMotor(MOTOR\_BOTH,127);
}
```

Kapitel 5

Konstanten

Die RNControl-Bibliothek stellt eine Reihe von Konstanten zur Verfügung,
die das Arbeiten erleichtern.

5.0.1 Konstanten

Konstante	**Wert**	**Anwendung**
USE_MOTOR	0/1	USE_MOTOR=0 alles rund um Motor wird nicht hinzu kompiliert die betroffenen Ports stehen zur freien Verfügung
ON	1	z.B zum aktivieren eines Ports
OFF	0	alternativ zu *HIGH/LOW*
FORWARD	0	Motordrehrichtung.
BACKWARD	1	Motordrehrichtung.
MOTOR_ONE	0	Konstante für den linken Motor
MOTOR_TWO	1	Konstante für den rechten Motor
MOTOR_BOTH	2	Konstante, wenn beide Motoren angesprochen werden sollen
MOTOR_DIR_FORWARD	0	
MOTOR_DIR_BACKWARD	1	
MOTOR_ARC_CW	0	Kreisbogen im Uhrzeigersinn
MOTOR_ARC_CCW	1	Kreisbogen entgegen dem Uhrzeigersinn
MOTOR_ARC_RADIUS	20	maximaler Radius
MAX_OFFSET	25	maximaler prozentualer Anteil von speed
MIN_OFFSET	-25	minimaler prozentualer Anteil von speed
MAX_SPEED	255	maximale Geschwindigkeit eines Motors

Tabelle 5.1: Konstanten

5.0.2 Deaktivierung sämtlicher Motorbefehle

Unter Umständen kann es notwendig sein, dass am RN-Control kein Motor angeschlossen werden muss. Sollte dies der Fall sein, kann die RN-Control Bibliothek durch setzen der Konstanten USE_MOTOR = 0 verändert werden. Wird die Konstante auf 0 gesetzt, werden keine Methoden mit kompiliert, dies spart Speicherplatz und die belegten Motor-Pins auf dem RN-Control Board sind frei und können beliebig genutzt werden.

Was ist zu tun?

1. Im Dateiexplorer das Verzeichnis öffnen indem sich die ARDUINO-Software befindet
2. Das Unterverzeichnis *libraries* öffnen
3. Das Unterverzeichnis *RNControl* öffnen
4. Die Datei *RNControl.h* in einem beliebigen Editor öffnen
5. Die Zeile
   #define USE_MOTOR 1
   korrigieren in
   #define USE_MOTOR 0
6. Die Datei als Textdatei speichern.
7. Das war's

Kapitel 6

Tasterabfrage

6.1 Methode getButton()

Das RNControl stellt fünf Taster zur Verfügung. Diese sind alle über den Analogport PortA Pin 7 mit dem ATMega32 verbunden. Die Abfrage diese Methode liefert die ID des Tasters zurück (1-5)

Befehl:
uint8_t getButton()

Anwendung:
Abfrage der Taster auf dem RNControl

Beispiel: (liefert die ID des Tasters zurück der gedrückt wurde.)
uint8_t taster = getButton();

Kapitel 7

Methode sound()

derzeit nicht implementiert

Kapitel 8

Serielle Schnittstelle RS232

Es können die Arduino-Befehle genutzt werden.

Kapitel 9

Timer

Es können die Arduino-Befehle genutzt werden. Der ATMega32 stellt drei Timer zur Verfügung

- Timer 0, 8Bit Timer
- Timer 1, 16Bit Timer
- Timer 2, 8Bit Timer

Kapitel 10

I2C

Es können die Arduino-Befehle bzw. die verfügbare Bibliothek genutzt werden.

Kapitel 11

Interrupts

Um Interrupts zur Verwenden, können die Arduino-Befehle genutzt werden.

Kapitel 12

ARDUINO - Core-Änderungen

?? Nachfolgende Änderungen wurden in den Core-Dateien von Arduino durchgeführt, um den ATMega32 nutzen zu können.

12.1 pins_arduino.h

Für die Verwendung des ATMega32 wurde diese Datei erstellt. Sie muss in das Verzeichnis mega32 kopiert werden. Siehe Installationsanweiseung.

12.2 board.txt

Nachfolgende Eintragungen wurden hinzugefügt:

```
atmega32.name=RN-Control/ATMega32
atmega32.upload.protocol=stk500v2
atmega32.upload.maximum_size=32768
atmega32.upload.speed=57600
atmega32.upload.using=stk500v2
atmega32.build.mcu=atmega32
atmega32.build.f_cpu=16000000L
atmega32.build.core=arduino
```

12.3 programmers.txt

Um das RN-Control mit einem USB ISP Programer nutzen zu können wurden nachfolgende Eintragungen in der Datei hinzugefügt

```
stk500v2.name=AVR stk500v2
stk500v2.communication=serial
stk500v2.protocol=stk500v2
```

Tabellenverzeichnis

Anhang A

Historie

Version	Kommentar
0.1	initiale Version
0.2	Konstanten MOTOR_ONE, MOTOR_TWO verwendet
	Befehl *arcMotors()* implementiert
	Befehl *setMotorOffset()* implementiert
0.3	Bugfix, Befehl *setMotorReverse()* implementiert
0.4	Befehl *motorExt()* implementiert
1.0	Update auf Arudino IDE 1.0.1 (PortIDs geändert)
1.1	Bugfixing

Tabelle A.1: Änderungshistorie

Anhang B

Komplette Portliste

Port	HWPin	ArduinoPin	Bedeutung
PORT A	40	D30	A0, Analog Input/Output
PORT A	39	D31	A1, Analog Input/Output
PORT A	38	D24	A2, Analog Input/Output
PORT A	37	D25	A3, Analog Input/Output
PORT A	36	D26	A4, Analog Input/Output
PORT A	35	D27	A5, Analog Input/Output
PORT A	34	D28	A6, Analog Input/Output
PORT A	33	D29	A7, Abfrage der Tasten auf dem RN-Control
PORT B	1	D8	PB0, Digital Input/Output, Vorbelegung: Motor 2, Kanal 1
PORT B	2	D9	PB1, Digital Input/Output Vorbelegung: Motor 2, Kanal 2
PORT B	3	D10	PB2, Digital Input/Output Vorbelegung: INT2, wird auch von I2C als Alarmleitung genutzt
PORT B	4	D11	PB3, Digital Input/Output OC, steht zur freien Verfügung
PORT B	5	D12	PB4, Digital Input/Output SS, steht zur freien Verfügung
PORT B	6	D13	PB5, Digital Input/Output Vorbelegung: MOSI, wird von ISP genutzt
PORT B	7	D14	PB6, Digital Input/Output Vorbelegung: MISO, wird von ISP genutzt
PORT B	8	D15	PB7, Digital Input/Output Vorbelegung: SCK, wird von ISP genutzt
PORT C	22	D16	PC0, Digital Input/Output, Vorbelegung: SCL, I2C
PORT C	23	D17	PC1, Digital Input/Output Vorbelegung: SDA, I2C

PORT C	24	D18	PC2, Digital Input/Output
			TCK, steht zur freien Verfügung
PORT C	25	D19	PC3, Digital Input/Output
			TMS, steht zur freien Verfügung
PORT C	26	D20	PC4, Digital Input/Output
			TDO, steht zur freien Verfügung
PORT C	27	D21	PC5, Digital Input/Output
			TDI, steht zur freien Verfügung
PORT C	28	D22	PC6, Digital Input/Output
			Vorbelegung: wird von Motor 1 Kanal 1 genutzt
PORT C	29	D23	PC7, Digital Input/Output
			Vorbelegung: wird von Motor 1 Kanal 2 genutzt
PORT D	14	D0	PD0, Digital Input/Output,
			Vorbelegung: RXD, RS232
PORT D	15	D1	PD1, Digital Input/Output
			Vorbelegung: TXD, RS232
PORT D	16	D2	PD2, Digital Input/Output
			INT0, steht zur freien Verfügung
PORT D	17	D3	PPD3, Digital Input/Output
			INT1, steht zur freien Verfügung
PORT D	18	D4	PD4, Digital Input/Output
			Vorbelegung: OC1B, PWM Motor 1
PORT D	19	D5	PD5, Digital Input/Output
			Vorbelegung: OC1A, PWM Motor 2
PORT D	20	D6	PD6, Digital Input/Output
			ICP, steht zur freien Verfügung
PORT D	21	D7	PD7, Digital Input/Output
			OC2, steht zur freien Verfügung

Tabelle B.1: tab:PortA-D