```
/*
  Ardumower (www.ardumower.de)
  Copyright (c) 2013-2015 by Alexander Grau
  Copyright (c) 2013-2015 by Sven Gennat
  Copyright (c) 2014 by Maxime Carpentieri
  Copyright (c) 2014-2015 by Stefan Manteuffel
  Copyright (c) 2015 by Uwe Zimprich

  Private-use only! (you need to ask for a commercial-use)

  This program is free software: you can redistribute it and/or
modify
  it under the terms of the GNU General Public License as published
by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/
licenses/>.

  Private-use only! (you need to ask for a commercial-use)

*/

/* Ardumower Chassis Kit 1.0 - robot configuration (Ardumower
electronics, Arduino Mega)
   http://wiki.ardumower.de/index.php?title=Ardumower_chassis

   Requires: Ardumower PCB v0.5  ( https://www.marotronics.de/
Ardumower-Board-Prototyp )

*/

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* *
// NOTE: Verify in config.h that you have enabled 'USE_MOWER' !
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* *

#include "config.h"
#ifdef USE_MOWER

#include <Arduino.h>
#include "mower.h"
#include "due.h"

// ------ pins--------------------------------------
#define pinMotorEnable  37         // EN motors enable
```

```
#define pinMotorLeftPWM 5          // M1_IN1 left motor PWM pin
#define pinMotorLeftDir 31         // M1_IN2 left motor Dir pin
#define pinMotorLeftSense A1       // M1_FB  left motor current
sense
#define pinMotorLeftFault 25       // M1_SF  left motor fault

#define pinMotorRightPWM  3        // M2_IN1 right motor PWM pin
#define pinMotorRightDir 33        // M2_IN2 right motor Dir pin
#define pinMotorRightSense A0      // M2_FB  right motor current
sense
#define pinMotorRightFault 27      // M2_SF  right motor fault

#define pinMotorMowPWM 2           // M1_IN1 mower motor PWM pin (if
using MOSFET, use this pin)
#define pinMotorMowDir 29          // M1_IN2 mower motor Dir pin (if
using MOSFET, keep unconnected)
#define pinMotorMowSense A3        // M1_FB  mower motor current
sense
#define pinMotorMowFault 26        // M1_SF  mower motor fault   (if
using MOSFET/L298N, keep unconnected)
#define pinMotorMowEnable 28       // EN mower motor enable      (if
using MOSFET/L298N, keep unconnected)
#define pinMotorMowRpm A11

#define pinBumperLeft 39           // bumper pins
#define pinBumperRight 38

#define pinDropLeft 45           // drop pins
Dropsensor - Absturzsensor
#define pinDropRight 23          // drop pins
Dropsensor - Absturzsensor

#define pinSonarCenterTrigger 24   // ultrasonic sensor pins
#define pinSonarCenterEcho 22
#define pinSonarRightTrigger 30
#define pinSonarRightEcho 32
#define pinSonarLeftTrigger 34
#define pinSonarLeftEcho 36
#define pinPerimeterRight A4       // perimeter
#define pinPerimeterLeft A5

#define pinLED 13                  // LED
#define pinBuzzer 53               // Buzzer
#define pinTilt 35                 // Tilt sensor (required for TC-
G158 board)
#define pinButton 51               // digital ON/OFF button
#define pinBatteryVoltage A2       // battery voltage sensor
#define pinBatterySwitch 4         // battery-OFF switch
#define pinChargeVoltage A9        // charging voltage sensor
#define pinChargeCurrent A8        // charge current sensor
#define pinChargeRelay 50          // charge relay
#define pinRemoteMow 12            // remote control mower motor
#define pinRemoteSteer 11          // remote control steering
#define pinRemoteSpeed 10          // remote control speed
```

```
#define pinRemoteSwitch 52         // remote control switch
#define pinVoltageMeasurement A7   // test pin for your own voltage
measurements
#ifdef __AVR__
  #define pinOdometryLeft A12      // left odometry sensor
  #define pinOdometryLeft2 A13     // left odometry sensor (optional
two-wire)
  #define pinOdometryRight A14     // right odometry sensor
  #define pinOdometryRight2 A15    // right odometry sensor
(optional two-wire)
#else
  #define pinOdometryLeft DAC0     // left odometry sensor
  #define pinOdometryLeft2 DAC1    // left odometry sensor (optional
two-wire)
  #define pinOdometryRight CANRX   // right odometry sensor
  #define pinOdometryRight2 CANTX  // right odometry sensor
(optional two-wire)
#endif
#define pinLawnFrontRecv 40        // lawn sensor front receive
#define pinLawnFrontSend 41        // lawn sensor front sender
#define pinLawnBackRecv 42         // lawn sensor back receive
#define pinLawnBackSend 43         // lawn sensor back sender
#define pinUserSwitch1 46          // user-defined switch 1
#define pinUserSwitch2 47          // user-defined switch 2
#define pinUserSwitch3 48          // user-defined switch 3
#define pinRain 44                 // rain sensor
// IMU (compass/gyro/accel): I2C  (SCL, SDA)
// Bluetooth: Serial2 (TX2, RX2)
// GPS: Serial3 (TX3, RX3)

// ------- baudrates-----------------------------------
#define BAUDRATE 19200            // serial output baud rate
#define PFOD_BAUDRATE 19200       // pfod app serial output baud
rate
#define PFOD_PIN 1234             // Bluetooth pin

//#define USE_DEVELOPER_TEST     1      // uncomment for new
perimeter signal test (developers)

Mower robot;


Mower::Mower(){
  name = "Ardumower";
  // ------- wheel motors -------------------------------
  motorAccel       = 500;  // motor wheel acceleration - only
functional when odometry is not in use (warning: do not set too low)
  motorSpeedMaxRpm       = 25;   // motor wheel max RPM (WARNING: do
not set too high, so there's still speed control when battery is
low!)
  motorSpeedMaxPwm    = 255;  // motor wheel max Pwm  (8-bit
PWM=255, 10-bit PWM=1023)
  motorPowerMax     = 75;    // motor wheel max power (Watt)
  motorSenseRightScale = 15.3; // motor right sense scale (mA=(ADC-
```

```
zero)/scale)
  motorSenseLeftScale = 15.3; // motor left sense scale  (mA=(ADC-
zero)/scale)
  motorPowerIgnoreTime = 2000; // time to ignore motor power (ms)
  motorZeroSettleTime   = 2000 ; // how long (ms) to wait for motors
to settle at zero speed
  motorRollTimeMax    = 3000;  // max. roll time (ms)
  motorReverseTime    = 3500;  // max. reverse time (ms)
  motorForwTimeMax   = 60000; // max. forward time (ms) / timeout
  motorBiDirSpeedRatio1 = 0.3;   // bidir mow pattern speed ratio 1
  motorBiDirSpeedRatio2 = 0.92;   // bidir mow pattern speed ratio 2
  // ---- normal control ---
  motorLeftPID.Kp       = 0.87;    // motor wheel PID controller
  motorLeftPID.Ki       = 0.29;
  motorLeftPID.Kd       = 0.25;
  /*// ---- fast control ---
  motorLeftPID.Kp       = 1.76;    // motor wheel PID controller
  motorLeftPID.Ki       = 0.87;
  motorLeftPID.Kd       = 0.4;*/

  motorRightSwapDir     = 0;    // inverse right motor direction?
  motorLeftSwapDir      = 1;    // inverse left motor direction?
  // ------ mower motor --------------------------------
  motorMowAccel       = 0.1;  // motor mower acceleration (warning:
do not set too high)
  motorMowSpeedMaxPwm   = 255;    // motor mower max PWM
  motorMowPowerMax = 75.0;     // motor mower max power (Watt)
  motorMowModulate  = 0;      // motor mower cutter modulation?
  motorMowRPMSet        = 3300;   // motor mower RPM (only for
cutter modulation)
  motorMowSenseScale = 15.3; // motor mower sense scale (mA=(ADC-
zero)/scale)
  motorMowPID.Kp = 0.005;    // motor mower RPM PID controller
  motorMowPID.Ki = 0.01;
  motorMowPID.Kd = 0.01;
  //  ------ bumper -----------------------------------
  bumperUse         = 0;      // has bumpers?
  //  ------ drop -------------------------------------
  dropUse          = 0;     // has drops?
Dropsensor - Absturzsensor vorhanden ?
  dropcontact      = 1;     //contact 0-openers 1-closers
Dropsensor - Kontakt 0-Öffner - 1-Schließer betätigt gegen GND
  // ------ rain -------------------------------------
  rainUse          = 0;      // use rain sensor?
  // ------ sonar ------------------------------------
  sonarUse          = 0;      // use ultra sonic sensor? (WARNING:
robot will slow down, if enabled but not connected!)
  sonarLeftUse      = 0;
  sonarRightUse     = 0;
  sonarCenterUse    = 0;
  sonarTriggerBelow = 900;    // ultrasonic sensor trigger distance
  // ------ perimeter --------------------------------
  perimeterUse       = 0;      // use perimeter?
  perimeterTriggerTimeout = 0;      // perimeter trigger timeout
```

```
when escaping from inside (ms)
  perimeterTrackRollTime  = 3000;   // perimter tracking roll time
(ms)
  perimeterTrackRevTime   = 2000;   // perimter tracking reverse
time (ms)
  perimeterPID.Kp    = 51.0;  // perimeter PID controller
  perimeterPID.Ki    = 12.5;
  perimeterPID.Kd    = 0.8;
  trackingPerimeterTransitionTimeOut = 2000;
  trackingErrorTimeOut = 10000;
  trackingBlockInnerWheelWhilePerimeterStruggling = 1;
  // ------ lawn sensor --------------------------------
  lawnSensorUse     = 0;       // use capacitive Sensor
  // ------  IMU (compass/accel/gyro) ----------------------
  imuUse            = 0;       // use IMU?
  imuCorrectDir     = 0;       // correct direction by compass?
  imuDirPID.Kp      = 5.0;     // direction PID controller
  imuDirPID.Ki      = 1.0;
  imuDirPID.Kd      = 1.0;
  imuRollPID.Kp     = 0.8;   // roll PID controller
  imuRollPID.Ki     = 21;
  imuRollPID.Kd     = 0;
  // ------ model R/C ----------------------------------
  remoteUse         = 0;       // use model remote control (R/C)?
  // ------ battery ------------------------------------
  batMonitor = 1;              // monitor battery and charge
voltage?
  batGoHomeIfBelow = 23.7;     // drive home voltage (Volt)
  batSwitchOffIfBelow = 21.7;  // switch off battery if below
voltage (Volt)
  batSwitchOffIfIdle = 1;      // switch off battery if idle
(minutes)
  batFactor       = 0.495;      // battery conversion factor  / 10
due to arduremote bug, can be removed after fixing (look in
robot.cpp)
  batChgFactor    = 0.495;      // battery conversion factor  / 10
due to arduremote bug, can be removed after fixing (look in
robot.cpp)
  batFull          =29.4;      // battery reference Voltage (fully
charged) PLEASE ADJUST IF USING A DIFFERENT BATTERY VOLTAGE! FOR a
12V SYSTEM TO 14.4V
  batChargingCurrentMax =1.6;  // maximum current your charger can
devliver
  batFullCurrent  = 0.3;      // current flowing when battery is
fully charged
  startChargingIfBelow = 27.0; // start charging if battery Voltage
is below
  chargingTimeout = 12600000; // safety timer for charging (ms)
12600000 = 3.5hrs
  // Sensorausgabe Konsole      (chgSelection =0)
  // Einstellungen ACS712 5A    (chgSelection =1   /   chgSenseZero
~ 511    /    chgFactor = 39    /    chgSense =185.0    /
chgChange = 0 oder 1    (je nach  Stromrichtung)   /   chgNull  = 2)
  // Einstellungen INA169 board (chgSelection =2)
```

```
  chgSelection    = 2;
  chgSenseZero    = 511;        // charge current sense zero point
  chgFactor       = 39;         // charge current conversion factor
- Empfindlichkeit nimmt mit ca. 39/V Vcc ab
  chgSense        = 185.0;      // mV/A empfindlichkeit des
Ladestromsensors in mV/A (Für ACS712 5A = 185)
  chgChange       = 0;          // Messwertumkehr von - nach +
1 oder 0
  chgNull         = 2;          // Nullduchgang abziehen (1 oder 2)
  // ------  charging station ----------------------------
  stationRevTime     = 4000;    // charge station reverse time (ms)
  stationRollTime    = 2000;    // charge station roll time (ms)
  stationForwTime    = 2000;    // charge station forward time (ms)
  stationCheckTime   = 2500;    // charge station reverse check time
(ms)
  // ------ odometry ------------------------------------
  odometryUse       = 1;       // use odometry?
  twoWayOdometrySensorUse = 0; // use optional two-wire odometry
sensor?
  odometryTicksPerRevolution = 1060;   // encoder ticks per one full
resolution
  odometryTicksPerCm = 13.49;  // encoder ticks per cm
  odometryWheelBaseCm = 36;    // wheel-to-wheel distance (cm)
  odometryRightSwapDir = 0;       // inverse right encoder
direction?
  odometryLeftSwapDir  = 1;       // inverse left encoder direction?
  // ----- GPS -----------------------------------------
  gpsUse            = 0;       // use GPS?
  // ----- other ---------------------------------------
  buttonUse         = 0;       // has digital ON/OFF button?
  // ----- user-defined switch --------------------------
  userSwitch1       = 0;       // user-defined switch 1 (default
value)
  userSwitch2       = 0;       // user-defined switch 2 (default
value)
  userSwitch3       = 0;       // user-defined switch 3 (default
value)
  // ----- timer ---------------------------------------
  timerUse          = 0;       // use RTC and timer?
  // ------ configuration end
----------------------------------------
}


// remote control (RC) ppm signal change interrupt
ISR(PCINT0_vect){
  unsigned long timeMicros = micros();
  boolean remoteSpeedState = digitalRead(pinRemoteSpeed);
  boolean remoteSteerState = digitalRead(pinRemoteSteer);
  boolean remoteMowState = digitalRead(pinRemoteMow);
  boolean remoteSwitchState = digitalRead(pinRemoteSwitch);
  robot.setRemotePPMState(timeMicros, remoteSpeedState,
remoteSteerState, remoteMowState, remoteSwitchState);
}
```

```
// odometry signal change interrupt
// mower motor speed sensor interrupt
// NOTE: when choosing a higher perimeter sample rate (38 kHz) and
using odometry interrupts,
// the Arduino Mega cannot handle all ADC interrupts anymore - the
result will be a 'noisy'
// perimeter filter output (mag value) which disappears when
disabling odometry interrupts.
// SOLUTION: allow odometry interrupt handler nesting (see odometry
interrupt function)
// http://www.nongnu.org/avr-libc/user-manual/
group__avr__interrupts.html
ISR(PCINT2_vect, ISR_NOBLOCK){
  unsigned long timeMicros = micros();
  boolean odometryLeftState = digitalRead(pinOdometryLeft);
  boolean odometryLeftState2 = digitalRead(pinOdometryLeft2);
  boolean odometryRightState = digitalRead(pinOdometryRight);
  boolean odometryRightState2 = digitalRead(pinOdometryRight2);
  boolean motorMowRpmState = digitalRead(pinMotorMowRpm);
  robot.setOdometryState(timeMicros, odometryLeftState,
odometryRightState, odometryLeftState2, odometryRightState2);
  robot.setMotorMowRPMState(motorMowRpmState);
}

// mower motor speed sensor interrupt
//void rpm_interrupt(){
//}


// WARNING: never use 'Serial' in the Ardumower code - use 'Console'
instead
// (required so we can use Arduino Due native port)

void Mower::setup(){
  Wire.begin();
  Console.begin(BAUDRATE);
  //while (!Console) ; // required if using Due native port
  Console.println("SETUP");
  rc.initSerial(PFOD_BAUDRATE);

  // http://sobisource.com/arduino-mega-pwm-pin-and-frequency-timer-
control/
  // http://www.atmel.com/images/doc2549.pdf
  #ifdef __AVR__
  // NOTE: next line commented out so we can use default 450 Hz PWM
freq (perimeter v2 otherwise uses the same freq band)
  //  TCCR3B = (TCCR3B & 0xF8) | 0x02;    // set PWM frequency 3.9
Khz (pin2,3,5)
  #endif

  // i2c -- turn off internal pull-ups (and use external pull-ups)
  //digitalWrite(SDA, 0);
  //digitalWrite(SCL, 0);
```

```
// keep battery switched ON
pinMode(pinBatterySwitch, OUTPUT);
digitalWrite(pinBatterySwitch, HIGH);

// LED, buzzer, battery
pinMode(pinLED, OUTPUT);
pinMode(pinBuzzer, OUTPUT);
digitalWrite(pinBuzzer,0);
pinMode(pinBatteryVoltage, INPUT);
pinMode(pinChargeCurrent, INPUT);
pinMode(pinChargeVoltage, INPUT);
pinMode(pinChargeRelay, OUTPUT);
setActuator(ACT_CHGRELAY, 0);

// left wheel motor
pinMode(pinMotorEnable, OUTPUT);
digitalWrite(pinMotorEnable, HIGH);
pinMode(pinMotorLeftPWM, OUTPUT);
pinMode(pinMotorLeftDir, OUTPUT);
pinMode(pinMotorLeftSense, INPUT);
pinMode(pinMotorLeftFault, INPUT);

// right wheel motor
pinMode(pinMotorRightPWM, OUTPUT);
pinMode(pinMotorRightDir, OUTPUT);
pinMode(pinMotorRightSense, INPUT);
pinMode(pinMotorRightFault, INPUT);

// mower motor
pinMode(pinMotorMowDir, OUTPUT);
pinMode(pinMotorMowPWM, OUTPUT);
pinMode(pinMotorMowSense, INPUT);
pinMode(pinMotorMowRpm, INPUT);
pinMode(pinMotorMowEnable, OUTPUT);
digitalWrite(pinMotorMowEnable, HIGH);
pinMode(pinMotorMowFault, INPUT);

// lawn sensor
pinMode(pinLawnBackRecv, INPUT);
pinMode(pinLawnBackSend, OUTPUT);
pinMode(pinLawnFrontRecv, INPUT);
pinMode(pinLawnFrontSend, OUTPUT);

// perimeter
pinMode(pinPerimeterRight, INPUT);
pinMode(pinPerimeterLeft, INPUT);

// button
pinMode(pinButton, INPUT);
pinMode(pinButton, INPUT_PULLUP);

// bumpers
pinMode(pinBumperLeft, INPUT);
```

```
  pinMode(pinBumperLeft, INPUT_PULLUP);
  pinMode(pinBumperRight, INPUT);
  pinMode(pinBumperRight, INPUT_PULLUP);

 // drops
  pinMode(pinDropLeft,
INPUT);
// Dropsensor - Absturzsensor - Deklariert als Eingang
  pinMode(pinDropLeft,
INPUT_PULLUP);
// Dropsensor - Absturzsensor - Intern Pullab Widerstand aktiviert
(Auslösung erfolgt gegen GND)
  pinMode(pinDropRight,
INPUT);
// Dropsensor - Absturzsensor - Deklariert als Eingang
  pinMode(pinDropRight,
INPUT_PULLUP);
// Dropsensor - Absturzsensor - Intern Pullab Widerstand aktiviert
(Auslösung erfolgt gegen GND)

  // sonar
  pinMode(pinSonarCenterTrigger, OUTPUT);
  pinMode(pinSonarCenterEcho, INPUT);
  pinMode(pinSonarLeftTrigger, OUTPUT);
  pinMode(pinSonarLeftEcho, INPUT);
  pinMode(pinSonarRightTrigger, OUTPUT);
  pinMode(pinSonarRightEcho, INPUT);

  // rain
  pinMode(pinRain, INPUT);

  // R/C
  pinMode(pinRemoteMow, INPUT);
  pinMode(pinRemoteSteer, INPUT);
  pinMode(pinRemoteSpeed, INPUT);
  pinMode(pinRemoteSwitch, INPUT);

  // odometry
  pinMode(pinOdometryLeft, INPUT_PULLUP);
  pinMode(pinOdometryLeft2, INPUT_PULLUP);
  pinMode(pinOdometryRight, INPUT_PULLUP);
  pinMode(pinOdometryRight2, INPUT_PULLUP);

  // user switches
  pinMode(pinUserSwitch1, OUTPUT);
  pinMode(pinUserSwitch2, OUTPUT);
  pinMode(pinUserSwitch3, OUTPUT);

  // other
  pinMode(pinVoltageMeasurement, INPUT);

  // enable interrupts
  #ifdef __AVR__
    // R/C
```

```
    PCICR |= (1<<PCIE0);
    PCMSK0 |= (1<<PCINT4);
    PCMSK0 |= (1<<PCINT5);
    PCMSK0 |= (1<<PCINT6);
    PCMSK0 |= (1<<PCINT1);

    // odometry
    PCICR |= (1<<PCIE2);
    PCMSK2 |= (1<<PCINT20);
    PCMSK2 |= (1<<PCINT21);
    PCMSK2 |= (1<<PCINT22);
    PCMSK2 |= (1<<PCINT23);

    // mower motor speed sensor interrupt
    //attachInterrupt(5, rpm_interrupt, CHANGE);
    PCMSK2 |= (1<<PCINT19);
  #else
    // Due interrupts
    attachInterrupt(pinOdometryLeft, PCINT2_vect, CHANGE);
    attachInterrupt(pinOdometryLeft2, PCINT2_vect, CHANGE);
    attachInterrupt(pinOdometryRight, PCINT2_vect, CHANGE);
    attachInterrupt(pinOdometryRight2, PCINT2_vect, CHANGE);

    attachInterrupt(pinRemoteSpeed, PCINT0_vect, CHANGE);
    attachInterrupt(pinRemoteSteer, PCINT0_vect, CHANGE);
    attachInterrupt(pinRemoteMow, PCINT0_vect, CHANGE);
    attachInterrupt(pinRemoteSwitch, PCINT0_vect, CHANGE);

    //attachInterrupt(pinMotorMowRpm, rpm_interrupt, CHANGE);
    attachInterrupt(pinMotorMowRpm, PCINT2_vect, CHANGE);
  #endif

  // ADC
  ADCMan.init();
  ADCMan.setCapture(pinChargeCurrent, 1, true);//Aktivierung des
LaddeStrom Pins beim ADC-Managers
  ADCMan.setCapture(pinMotorMowSense, 1, true);
  ADCMan.setCapture(pinMotorLeftSense, 1, true);
  ADCMan.setCapture(pinMotorRightSense, 1, true);
  ADCMan.setCapture(pinBatteryVoltage, 1, false);
  ADCMan.setCapture(pinChargeVoltage, 1, false);
  ADCMan.setCapture(pinVoltageMeasurement, 1, false);
  perimeter.setPins(pinPerimeterLeft, pinPerimeterRight);

  imu.init(pinBuzzer);
  gps.init();

  Robot::setup();
}

void checkMotorFault(){
  if (digitalRead(pinMotorLeftFault)==LOW){
    robot.addErrorCounter(ERR_MOTOR_LEFT);
    robot.setNextState(STATE_ERROR, 0);
```

```
    //digitalWrite(pinMotorEnable, LOW);
    //digitalWrite(pinMotorEnable, HIGH);
  }
  if  (digitalRead(pinMotorRightFault)==LOW){
    robot.addErrorCounter(ERR_MOTOR_RIGHT);
    robot.setNextState(STATE_ERROR, 0);
    //digitalWrite(pinMotorEnable, LOW);
    //digitalWrite(pinMotorEnable, HIGH);
  }
  if (digitalRead(pinMotorMowFault)==LOW){
    robot.addErrorCounter(ERR_MOTOR_MOW);
    robot.setNextState(STATE_ERROR, 0);
    //digitalWrite(pinMotorMowEnable, LOW);
    //digitalWrite(pinMotorMowEnable, HIGH);
  }
}


int Mower::readSensor(char type){
  switch (type) {
//
motors------------------------------------------------------------------
---------------------------------
    case SEN_MOTOR_MOW: return ADCMan.read(pinMotorMowSense); break;
    case SEN_MOTOR_RIGHT: checkMotorFault(); return
ADCMan.read(pinMotorRightSense); break;
    case SEN_MOTOR_LEFT:  checkMotorFault(); return
ADCMan.read(pinMotorLeftSense); break;
    //case SEN_MOTOR_MOW_RPM: break; // not used - rpm is upated via
interrupt

//
perimeter---------------------------------------------------------------
---------------------------------
    case SEN_PERIM_LEFT: return perimeter.getMagnitude(0); break;
    //case SEN_PERIM_RIGHT: return Perimeter.getMagnitude(1); break;

//
battery-----------------------------------------------------------------
---------------------------------
    case SEN_BAT_VOLTAGE: ADCMan.read(pinVoltageMeasurement);
return ADCMan.read(pinBatteryVoltage); break;
    case SEN_CHG_VOLTAGE: return ADCMan.read(pinChargeVoltage);
break;
    //case SEN_CHG_VOLTAGE: return((int)
(((double)analogRead(pinChargeVoltage)) * batFactor)); break;
    case SEN_CHG_CURRENT: return ADCMan.read(pinChargeCurrent);
break;

//
buttons-----------------------------------------------------------------
---------------------------------
    case SEN_BUTTON: return(digitalRead(pinButton)); break;
```

```
//
bumper-------------------------------------------------------------
-----------------------------------
    case SEN_BUMPER_RIGHT: return(digitalRead(pinBumperRight));
break;
    case SEN_BUMPER_LEFT: return(digitalRead(pinBumperLeft)); break;

//
drop---------------------------------------------------------------
-----------------------------------
    case SEN_DROP_RIGHT: return(digitalRead(pinDropRight));
break;
// Dropsensor - Absturzsensor
    case SEN_DROP_LEFT: return(digitalRead(pinDropLeft));
break;
// Dropsensor - Absturzsensor

//
sonar--------------------------------------------------------------
-----------------------------------
    //case SEN_SONAR_CENTER: return(readURM37(pinSonarCenterTrigger,
pinSonarCenterEcho)); break;
    case SEN_SONAR_CENTER: return(readHCSR04(pinSonarCenterTrigger,
pinSonarCenterEcho)); break;
    case SEN_SONAR_LEFT: return(readHCSR04(pinSonarLeftTrigger,
pinSonarLeftEcho)); break;
    case SEN_SONAR_RIGHT: return(readHCSR04(pinSonarRightTrigger,
pinSonarRightEcho)); break;
   // case SEN_LAWN_FRONT:
return(measureLawnCapacity(pinLawnFrontSend, pinLawnFrontRecv));
break;
    //case SEN_LAWN_BACK:
return(measureLawnCapacity(pinLawnBackSend, pinLawnBackRecv));
break;

//
imu----------------------------------------------------------------
-----------------------------------
    //case SEN_IMU: imuYaw=imu.ypr.yaw; imuPitch=imu.ypr.pitch;
imuRoll=imu.ypr.roll; break;
//
rtc----------------------------------------------------------------
-----------------------------------
    case SEN_RTC:
      if (!readDS1307(datetime)) {
        Console.println("RTC data error!");
        addErrorCounter(ERR_RTC_DATA);
        setNextState(STATE_ERROR, 0);
      }
      break;
//
rain---------------------------------------------------------------
-----------------------------------
    case SEN_RAIN: if (digitalRead(pinRain)==LOW) return 1; break;
```

```
  }
  return 0;
}

void Mower::setActuator(char type, int value){
  switch (type){
    case ACT_MOTOR_MOW: setMC33926(pinMotorMowDir, pinMotorMowPWM,
value); break;// Motortreiber einstellung - bei Bedarf ändern z.B
setL298N auf setMC33926
    case ACT_MOTOR_LEFT: setMC33926(pinMotorLeftDir,
pinMotorLeftPWM, value); break;//
Motortreiber einstellung - bei Bedarf ändern z.B setL298N auf
setMC33926
    case ACT_MOTOR_RIGHT: setMC33926(pinMotorRightDir,
pinMotorRightPWM, value); break; //
Motortreiber einstellung - bei Bedarf ändern z.B setL298N auf
setMC33926
    case ACT_BUZZER: if (value == 0) noTone(pinBuzzer); else
tone(pinBuzzer, value); break;
    case ACT_LED: digitalWrite(pinLED, value); break;
    case ACT_USER_SW1: digitalWrite(pinUserSwitch1, value); break;
    case ACT_USER_SW2: digitalWrite(pinUserSwitch2, value); break;
    case ACT_USER_SW3: digitalWrite(pinUserSwitch3, value); break;
    case ACT_RTC:
      if (!setDS1307(datetime)) {
        Console.println("RTC comm error!");
        addErrorCounter(ERR_RTC_COMM);
        setNextState(STATE_ERROR, 0);
      }
      break;
    case ACT_CHGRELAY: digitalWrite(pinChargeRelay, value); break;
    //case ACT_CHGRELAY: digitalWrite(pinChargeRelay, !value);
break;
    case ACT_BATTERY_SW: digitalWrite(pinBatterySwitch, value);
break;
  }
}

void Mower::configureBluetooth(boolean quick){
  BluetoothConfig bt;
  bt.setParams(name, PFOD_PIN, PFOD_BAUDRATE, quick);
}

#endif
```