Fehlerbeschreibung: Der Mähroboter fährt beim automatischen Docking (Akku leer) in die Ladekontakte, startet aber sofort wieder den Mähvorgang bei aktiviertem „DOCK_AUTO_START" (insbesondere bei Verwendung von Ladegeräten, die den Ladestrom erst langsam ansteigen lassen).
Ursache: Die Variable „chargingCompleted" kann, wenn zum Zeitpunkt der Abfrage gerade die Bedingung „chargerConnectedState" erfüllt ist, aber der Ladestrom noch nicht angestiegen ist, schon „TRUE" sein, wodurch dann in der „robot.cpp" die Bedingung „battery.chargingHasCompleted()" erfüllt ist und der Mähvorgang automatisch fortgesetzt wird.
Abhilfe: Wenn der Mäher nicht an den Ladekontakten ist („chargerConnectedState" = FALSE), wird die Variable „chargingCompleted" ebenfalls auf FALSE gesetzt. Wechselt „chargerConnectedState" auf TRUE, so wird erst eine gewisse Zeit gewartet, bis auf den Zustand des Ladestroms und der Ladespannung geschaut wird.

Die gelb hinterlegten Codeteile sind in der „battery.h" einzufügen:

```
class Battery {
  public:
    bool batMonitor;
    float batGoHomeIfBelow;
    float batFullVoltage;
    float batSwitchOffIfBelow;  // switch off battery if below voltage (Volt)
    int batSwitchOffIfIdle;      // switch off battery if idle (minutes)
    int enableChargingTimeout;
    float batFullCurrent;
    float batteryVoltage;   // volts
    float chargingVoltage;  // volts
    float chargingCurrent;  // amps
    bool chargingEnabled;
    int chargingCompletedDelay; // ensure that loadingcurrent or loadingvoltage triggers
"chargingCompleted" condition for a longer periode
    bool chargingCompleted;
    void begin();
    void run();
    bool chargerConnected();
    void enableCharging(bool flag);
    bool shouldGoHome();
    bool chargingHasCompleted();
    bool underVoltage();
    void resetIdle();
    void switchOff();
  protected:
    unsigned long nextBatteryTime ;
    bool switchOffByOperator;
    unsigned long timeMinutes;
    bool chargerConnectedState;
    bool switchOffAllowedUndervoltage;
    bool switchOffAllowedIdle;
    unsigned long switchOffTime;
    unsigned long chargingStartTime;
    unsigned long nextCheckTime;
    unsigned long nextEnableTime;
    unsigned long nextPrintTime;
};
```

Die gelb hinterlegten Codeteile sind in der „battery.cpp" einzufügen:

```
void Battery::run(){
  if (millis() < nextBatteryTime) return;
  nextBatteryTime = millis() + 50;

  float voltage = batteryDriver.getChargeVoltage();
  if (abs(chargingVoltage-voltage) > 10) chargingVoltage = voltage;
  chargingVoltage = 0.9 * chargingVoltage + 0.1* voltage;

  voltage = batteryDriver.getBatteryVoltage();
  if (abs(batteryVoltage-voltage) > 10) batteryVoltage = voltage;
  float w = 0.995;
  if (chargerConnectedState) w = 0.9;
  batteryVoltage = w * batteryVoltage + (1-w) * voltage;

  chargingCurrent = 0.9 * chargingCurrent + 0.1 * batteryDriver.getChargeCurrent();

  if (!chargerConnectedState){
    if (chargingVoltage > 5){
      chargerConnectedState = true;
     DEBUGLN(F("CHARGER CONNECTED"));
      buzzer.sound(SND_OVERCURRENT, true);
    }
  }

  if (millis() >= nextCheckTime){
    nextCheckTime = millis() + 5000;
    if (chargerConnectedState){
      if (chargingVoltage <= 5){
        chargerConnectedState = false;
        nextEnableTime = millis() + 5000;     // reset charging enable time
        DEBUGLN(F("CHARGER DISCONNECTED"));
      }
    }
    timeMinutes = (millis()-chargingStartTime) / 1000 /60;
    if (underVoltage()) {
      DEBUGLN(F("SWITCHING OFF (undervoltage)"));
      buzzer.sound(SND_OVERCURRENT, true);
      if (switchOffAllowedUndervoltage)  batteryDriver.keepPowerOn(false);
    } else if ((millis() >= switchOffTime) || (switchOffByOperator)) {
      DEBUGLN(F("SWITCHING OFF (idle timeout)"));
      buzzer.sound(SND_OVERCURRENT, true);
      if ((switchOffAllowedIdle) || (switchOffByOperator)) batteryDriver.keepPowerOn(false);
    } else batteryDriver.keepPowerOn(true);

    if (millis() >= nextPrintTime){
      nextPrintTime = millis() + 60000;
      //print();
      /*DEBUG(F("charger conn="));
      DEBUG(chargerConnected());
      DEBUG(F(" chgEnabled="));
      DEBUG(chargingEnabled);
      DEBUG(F(" chgTime="));
      DEBUG(timeMinutes);
      DEBUG(F(" charger: "));
      DEBUG(chargingVoltage);
      DEBUG(F(" V  "));
      DEBUG(chargingCurrent);
      DEBUG(F(" A "));
      DEBUG(F(" bat: "));
      DEBUG(batteryVoltage);
      DEBUG(F(" V  "));
      DEBUG(F("switchOffAllowed="));
      DEBUG(switchOffAllowed);
      DEBUGLN();      */
    }
  }

  if (millis() > nextEnableTime){
    nextEnableTime = millis() + 5000;
    if (chargerConnectedState){
        // charger in connected state
        if (chargingEnabled){
          //if ((timeMinutes > 180) || (chargingCurrent < batFullCurrent)) {
          if (chargingCompletedDelay > 5) {  // chargingCompleted check first after 6 * 5000ms
= 30sec.
            chargingCompleted = ((chargingCurrent <= batFullCurrent) || (batteryVoltage >=
batFullVoltage));
          }
          else {
            chargingCompletedDelay++;
          }

          if (chargingCompleted) {
            // stop charging
            nextEnableTime = millis() + 1000 * enableChargingTimeout;   // check charging
current again in 30 minutes
            chargingCompleted = true;
            enableCharging(false);
          }
        } else {
           //if (batteryVoltage < startChargingIfBelow) {
              // start charging
              enableCharging(true);
              chargingStartTime = millis();
          //}
        }
    }
    else {
      // reset to avoid direct undocking after docking
      chargingCompleted        = false;
      chargingCompletedDelay   = 0;   // reset chargingCompleteted delay counter
    }

  }
}
```