Internet of Things + Conrad Adventskalender
- Kalenderwettbewerb -

IoT – Quadrocopter Landing Pad

Beitrag von: Paul Kaczmarczyk

Inhaltsverzeichnis

Projektbeschreibung

An Weihnachten erhielt ich einen Quadrocopter als Geschenk von meinem Opa. Ich hatte so viel Freude damit, dass ich mir gleich ein zweites Modell vom Typ Blade Inductrix zulegte. Das Fliegen mit dem Quadrocopter macht allerdings besonders viel Spaß, wenn man Aufgaben zu erfüllen hat.

Spielidee

Für die zu bewältigenden Aufgaben kam mir der Gedanke das Landing Pad zu bauen, bei dem die Aufgabe darin besteht durch einen Ring zu fliegen, ohne hängen zu bleiben und dann zu landen, sofern man eine Landefreigabe erhält. Wird die Landefreigabe vom Tower nicht gegeben, so muss man schweben, bis die Landefreigabe erteilt wird. Wird ohne Landefreigabe gelandet, erhält man als Strafe für die Landung einen Punktabzug. Landet man mit Freigabe erhält man Punkte auf sein Konto. Ein Hängenbleiben am Ring soll auch einen Punktabzug zur Folge haben. Um sich mit anderen zu messen, sollen die erreichten Punkte auch in einer Internet-weiten Highscore-Liste namentlich geführt werden.

Abbildung 1: Blade Inductrix Quadrocopter

Dieses Geschicklichkeitsspiel soll rundenbasiert ablaufen, wobei die Rundenzeit 60 s beträgt. Läuft die Runde ab, werden die addierten Punkte in die Highscore-Liste eingetragen. Der Start einer neuen Runde soll durch Druck auf einen Taster ermöglicht werden.

Grundaufbau der Hardware

Die Hardware besteht im Grundaufbau aus einer Schachtel aus dem Supermarkt, in der vorher Lebensmittel präsentiert wurden. Diese Schachtel bildet die Grundlage für alle weiteren Aufbauten. Hinzu kommt ein aus einer weiteren Wellpapierschachtel hergestelltes Stück, welches den zu durchfliegenden Ring bildet. In dieses Wellpapierstück ist ein kreisrunder Ausschnitt mit dem Durchmesser 200 mm hinein geschnitten worden. Das Ringteil ist frei schwingend aufgehängt. Als Achse wurde hierfür ein Schaschlikstäbchen verwendet, welches zwischen die Wellen der Wellpappe geschoben wurde.
Das Ausschwingen des Ringteils ist in eine Richtung bis zum Anschlag, der durch den Ausschnitt im Grundaufbau begrenzt wird, möglich. In die andere Richtung wird der Ausschlag durch Formschluss bei der Anlage an einer Tischkante verhindert. Dies ist nötig, damit das Ringteil beim

Schweben des Quadrocopters über der Landefläche (z.B. beim Warten auf die Landefreigabe) nicht durch den entstehenden Abwind ausschlägt. So werden falsche Ausschläge vermieden.

Abbildung 2: Quadrocopter Landing Pad-Ringteil

Abbildung 3: Quadrocopter Landing Pad-Landefläche

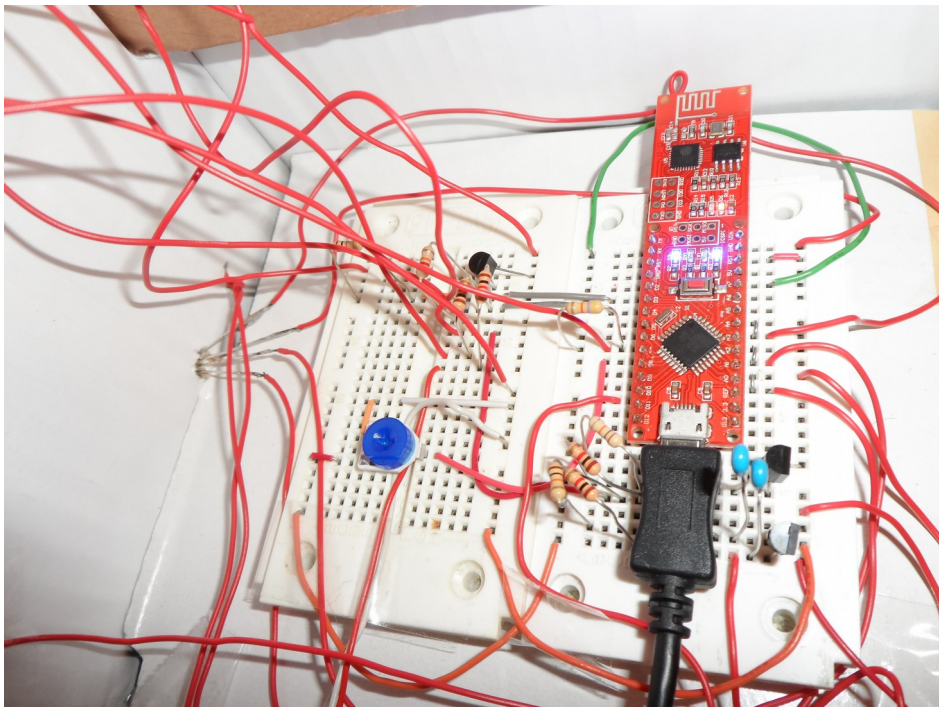
Abbildung 4: Elektronik

Elektronischer Schaltungsentwurf

Das zentrale Ziel beim Entwurf der Schaltungen war die Notwendigkeit sich auf die zur Verfügung stehenden Bauteile zu beschränken. Dies waren die im IoT-Adventskalender und Conrad Elektronik-Adventskalender verwendeten Bauteile. Die Verwendung weiterer elektronischer Bauteile wurde strikt abgelehnt, ist jedoch für weitere Entwicklungen außerhalb des Kalenderwettbewerbs wünschenswert und sinnvoll.
Die zentrale Instanz dieses Projekts wird vom Pretzelboard gebildet, welches eine Kombination eines Arduino Nano mit einem ESP8266 ist.

Sensorik

Die einzelnen Sensoren sollen im Folgenden näher beschrieben werden.

Einsatz des Neigungssensors

Der Neigungssensor aus dem Conrad Elektronik-Adventskalender wird zur Erkennung eines Anstoßes des Ringteils durch den Quadrocopter verwendet. Ähnlich wie bei der THT-Technik wird der Neigungssensor dazu durch die Wellpappe durchgesteckt und die Kontakte auf der Rückseite verbunden. Durch die mittlere Festigkeit der Wellpappe kann so auch später der Auslösewinkel des Neigungssensors angepasst werden. Dadurch kann der Schwierigkeitsgrad angepasst werden. Entweder der Sensor schließt früher, dann wird schon bei kleinen Berührungen ein Warnton ausgelöst und Punkte abgezogen, oder der Sensor schließt später, d.h. bei größeren Auslenkungswinkeln, so dass größere Fehler des Quadrocopter-Piloten toleriert werden.

Abbildung 5: Neigungssensor am Ringteil

Der Warnton, der beim Anecken gegeben werden soll, wird ohne Einsatz von Software durch einen astabilen Multivibrator erzeugt. Die dafür erforderlichen Transistoren sind im Conrad Elektronik-Adventskalender enthalten. Auch ein Piezo-Schallgeber findet sich unter den Bauteilen.

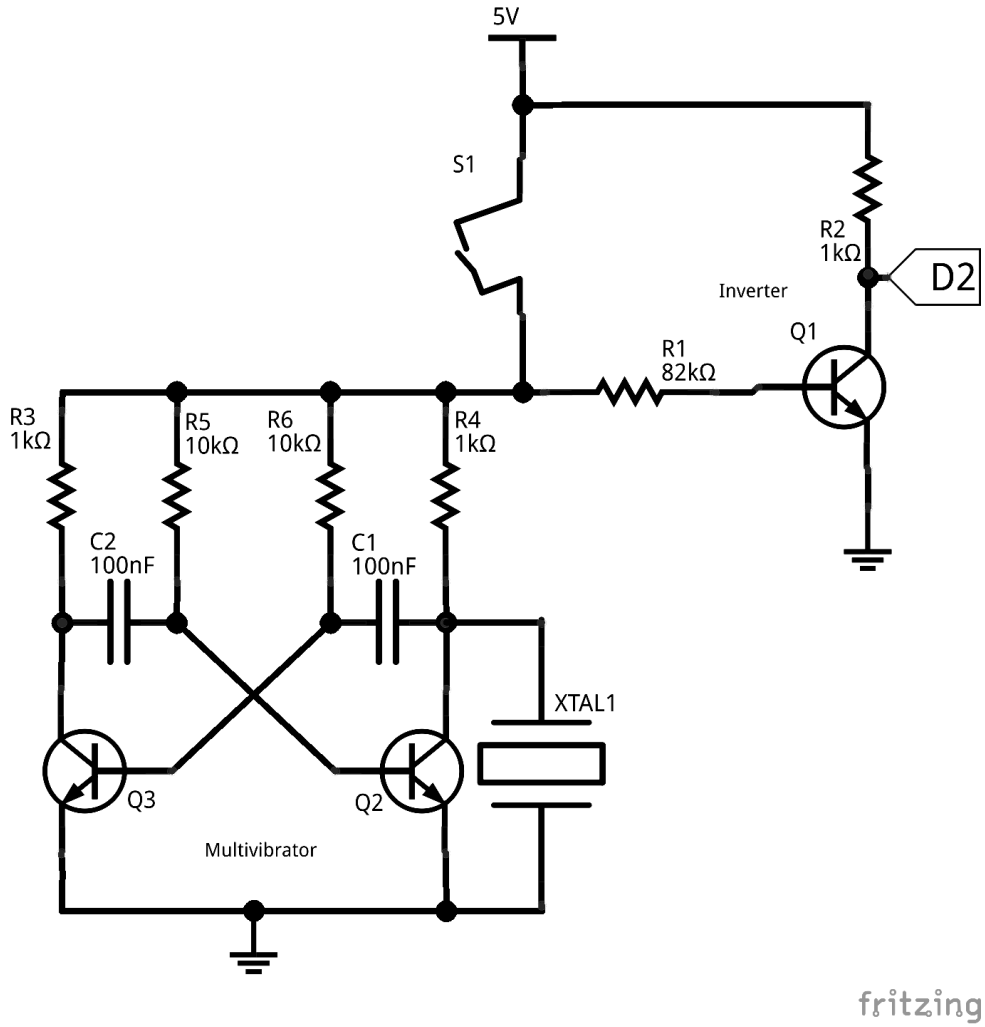

Abbildung 6: Schaltplan Neigungssensor

Erklärung der Schaltung

Schließt der Neigungssensor S1, so fließt ein Strom von der 5V-Stromversorgung des Arduino über den Schalter. Am dahinter liegenden Knotenpunkt teilt sich der Strom in zwei Teile.
Ein Teil fließt über den 82 kΩ Basiswiderstand zum Transistor Q1, welcher in Emitterschaltung als Schalter betrieben wird. Diese Schaltung dient als digitaler Inverter. Der digitale Wert der Leitung am Schalter wird also invertiert. Der invertierte Wert wird dann an Pin D2 des NanoESP-Boards angelegt und dort ausgelesen.
Der zweite Teil speist einen astabilen Multivibrator der für die Erzeugung eines Tons an XTAL1, einem Piezo-Schallwandler, verantwortlich ist. Der astabile Multivibrator ist symmetrisch aufgebaut. Es ergibt sich also folgende Periodendauer T:

$$t_1 = \ln 2 \cdot R_5 \cdot C_2$$

$$t_2 = \ln 2 \cdot R_6 \cdot C_1$$

$$T = t_1 + t_2$$

Mit $R_5 = R_6$ und $C_1 = C_2$ ergibt sich

$$t_1 = t_2$$

und damit

$$T = 2 \cdot t_1$$
$$= 2 \cdot \ln 2 \cdot R_5 \cdot C_2$$
$$T = 2 \cdot \ln 2 \cdot 10k\Omega \cdot 100nF$$

$$T = 1380\ \mu s$$

Damit ergibt sich eine Frequenz von

$$f = 1/T$$
$$= 724\ Hz$$

Einsatz des Fototransistors

Der Fototransistor aus dem IoT-Kalender wird zur Erkennung der Landung eingesetzt.
Da der Inductrix kleine LED-Scheinwerfer besitzt, können diese recht gut zur Erkennung der Landung mittels eines Fototransistors verwendet werden.

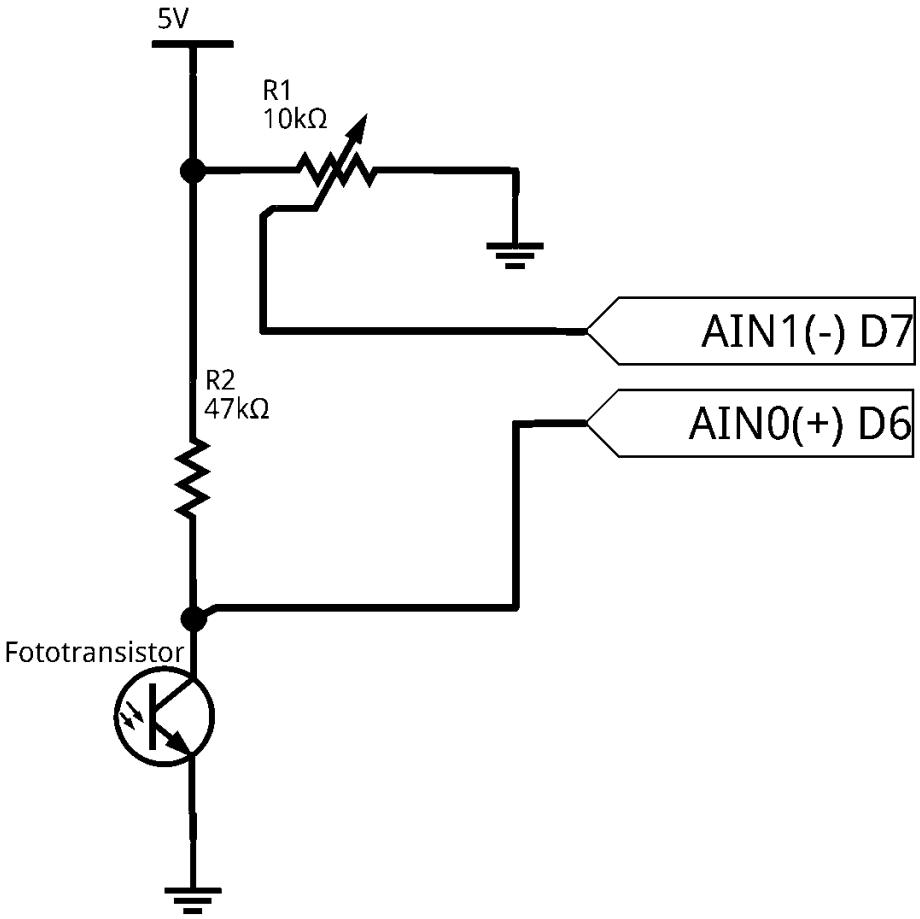

Abbildung 7: Landeerkennung

Abbildung 8: Der Fototransistor zur Landeerkennung

Erklärung der Schaltung

Der Fototransistor ist über einen 47 kΩ Vorwiderstand an der 5V Spannungsversorgung des Arduino angeschlossen. Sein Kollektorpotential, welches ja vom Lichteinfall abhängig ist, wird an den Pin D6 des Arduino geführt, welcher den Eingang AIN0, des integrierten analogen Komparators enthält. Liegt jetzt das Potential des Kollektors über einem Schwellwert, der über den Pin AIN1 festgelegt wird, wird der Ausgang des Komparators zu logisch 1, liegt er darunter wird er zu logisch 0. Diese Zustände können nun von der Software unmittelbar eingelesen werden. Der Wert des Komparatorausgangs befindet sich im Register ACSR in Bit AC0. Der Wert kann also mit ACSR & (1<<ACO) geprüft werden. Der Schwellwert bei dem eine Landung erkannt wird, wird mittels eines 10 kΩ Potentiometers, welches ebenfalls im Adventskalender enthalten war, eingestellt. Zum Einstellen stellt die Software ein Hilfsmittel zur Verfügung. Wird beim Einschalten während der Initialisierungsroutine der Startknopf gedrückt gelassen, wird über den Blau-Kanal der RGB-LED angezeigt, ob gerade eine Landung erkannt wird oder nicht. So kann der Spieler die Empfindlichkeit des Systems selbst einstellen. Wird der Startknopf losgelassen, fährt das System mit der normalen Initialisierung fort.

Signalisation

Im Folgenden soll die Signalisation näher beschrieben werden.

Landefreigabe

Die Landefreigabe, die in der Realität von einer Flugverkehrskontrollstelle auf einem Tower kommt, wird in diesem Projekt durch eine RGB-LED signalisiert.

Dabei gibt es folgende Zustände:

Farbe	Bedeutung
rot	Keine Landefreigabe erteilt, bei Landung Strafe: Punktabzug: -100 Punkte
grün	Landefreigabe erteilt, bei Landung 100 Punkte
blau	Abflugfreigabe nach eigenem Ermessen, Landung gäbe 0 Punkte

Der Zustand blau wird dabei 5 s nach einer erkannten Landung gehalten, damit der Spieler genug Zeit zum Wiederabflug hat. Danach wird in rot geschaltet. Ein vermasselter Wiederabflug ergibt somit folgerichtig auch 100 Punkte Abzug. Im der realen Fliegerei ist die benutzte Start-/Landefläche auch schnellstmöglich wieder frei zu machen. Alle 8 s wird zwischen rot und grün gewechselt, wenn keine Landung erfolgt. Das soll den Verkehr simulieren, der an dem Flughafen abgefertigt wird.

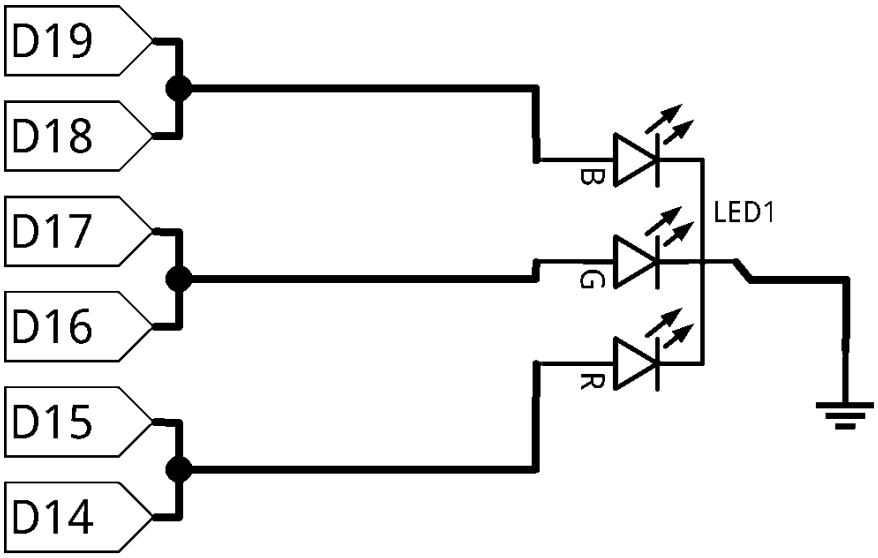

Abbildung 9: RGB-LED zur Landefreigabe

Erklärung der Schaltung

Da für den Wettbewerb nur die Bauteile aus den Kalendern verwendet werden durften und nicht genügend Vorwiderstände zur Verfügung standen, kam die Idee auf die internen Pull-Up-Widerstände des Arduino als Vorwiderstände zu verwenden. Das Datenblatt des Atmega328 gibt min. 20 kΩ pro Pin als internen Pull-Up-Widerstand an. Das ist zu hoch um eine angemessene Helligkeit der LED zu erreichen. Daher wurden jeweils zwei Pins zusammen geschaltet, so dass sich implizit eine Parallelschaltung der internen Pull-Up-Widerstände ergibt. Wichtig ist bei dem Verfahren, dass tatsächlich immer zwei Pins gleichzeitig geschaltet werden, da sich sonst ein Stromfluss von einem zum anderen Pin ergeben kann.
Für diese Nutzung der RGB-LED ist eine Pulsbreitenmodulation nicht erforderlich, da nur die drei Zustände „Landung frei“, „Landung nicht frei“ und „Abflug nach eigenem Ermessen“ gegeben werden sollen.

Positionslichter

Die Positionslichter sind den Positionslichtern realer Luftfahrzeuge nachempfunden. Diese sind auch links rot, rechts grün und hinten weiß. Dieses Schema wurde hier übernommen, wobei die Positionslichter zur Anzeige einer laufenden Runde verwendet werden, in welcher sie schwellend über eine Pulsbreitenmodulation ein und ausgeschaltet werden. Alle Positionslichter werden in gleichem Maße gedimmt, so dass alle am gleichen Pin D3 angeschlossen sind.

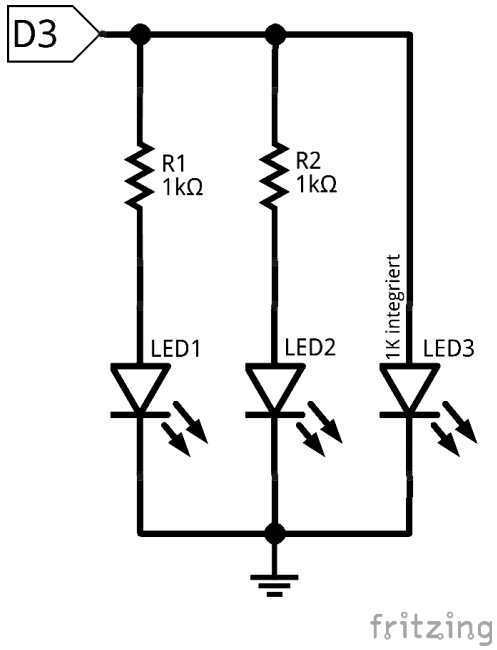

Abbildung 10: Positionslichter

Software

Die Software wurde unter objektorientierten Gesichtspunkten entwickelt. Dazu wurden Teile des Programms in die Klassen CGameEngine und CLandingClearance gepackt.
In CGameEngine wird die Spiellogik verwaltet. In CLandingClearance wird die Landefreigabenverwaltung durchgeführt.
Darüber hinaus wurde die Software in mehrere Dateien zerlegt und das Projekt leichter handeln zu können.

Die einzelnen Dateien des Projekts sind:

```
CGameEngine.h
CLandingClearance.h
espUtil.h
Quadrocopter_LandingPad.ino
rgb.h
```

Die ESP8266-spezifischen Routinen und der Aufbau des Webservers sind größtenteils an den IoT-Adventskalender angelehnt und werden hier nicht weiter behandelt. Über den Webserver auf dem Arduino ist es möglich mittels eines Webbrowsers z.B. auf einem Computer den Spielernamen fesetzulegen und über ein iframe wird die Highscore-Liste aus dem Internet angezeigt.
Desweiteren ist zu sagen, dass der DEBUG-Modus im vorliegenden Programm nicht mit if(debug) sondern mit #ifdef DEBUG abgefragt wird, so dass die DEBUG-Anweisungen schon durch den Präprozessor aussortiert werden und nicht kompiliert werden, was zur Performanceerhöhung

beiträgt.

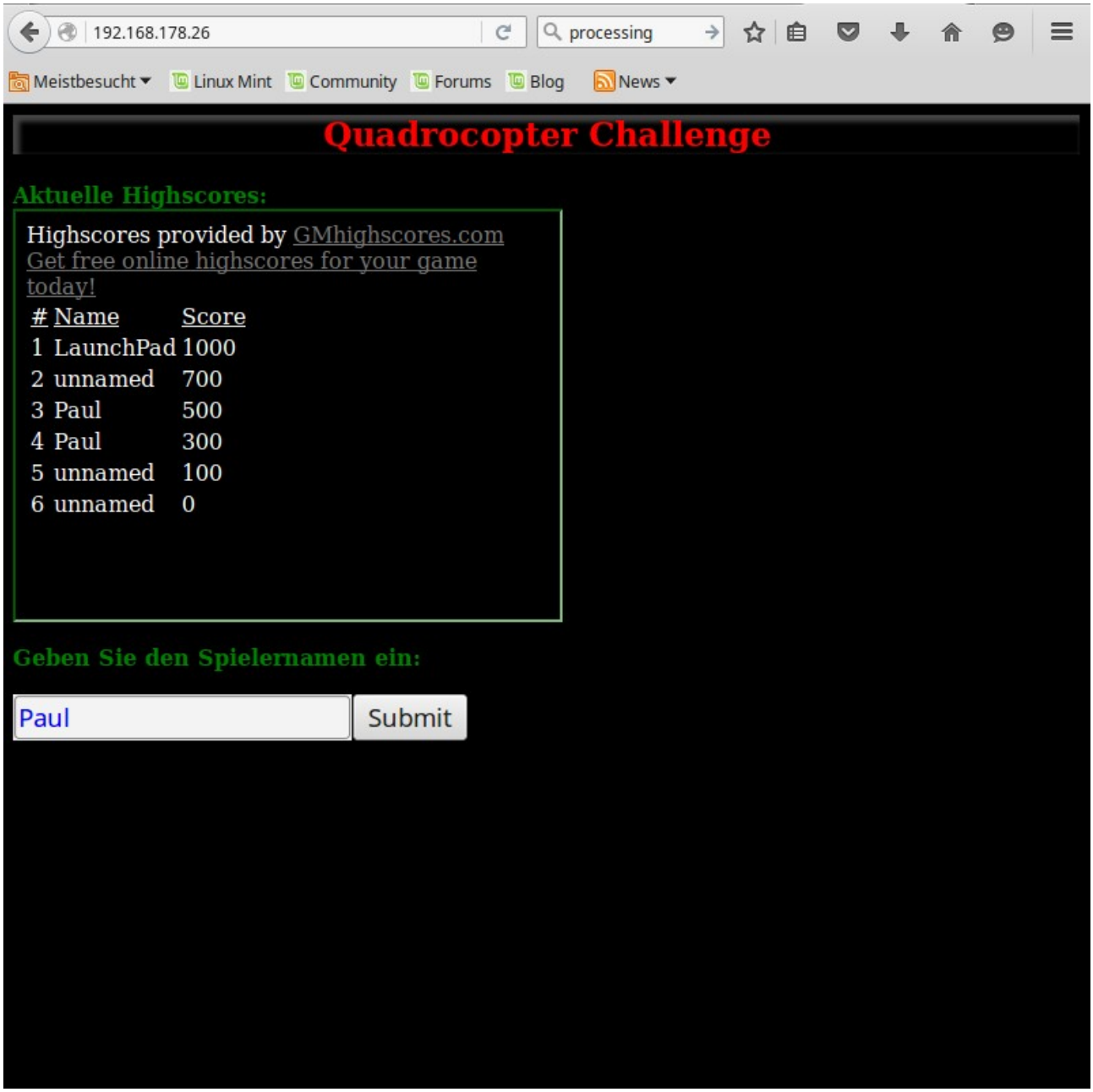

Abbildung 11: Website zur Eingabe des Namens und Anzeige der Highscores

Versand des Highscores

Lediglich das Versenden des Highscores soll kurz beleuchtet werden. Für die Highscores wurde der Online-Highscore-Dienst „Gamemaker Highscores“ gmhighscores.com gewählt. Dieser Dienst bietet die Online-Verwaltung von Highscores an. Leider ist es so, dass eine DLL (Dynamic Link Library) von gmhighscores zur Verfügung gestellt wird, die für die Übermittlung aus Spielen heraus verwendet werden soll. Diese DLL ist auf dem Arduino nicht zum Laufen zu bringen. Daher ist es erforderlich das Verhalten der DLL nachzubilden. Unter der Adresse
https://github.com/lord/gmhighscores/blob/master/newhighscore_action.php
lässt sich der PHP-Code von gmhighscores.com einsehen:

```
<?php
```

```
if (!isset($_GET['score'])) {
die("S");
}
if (!isset($_GET['user'])) {
die("U");
}
if (!isset($_GET['verify'])) {
die("V");
}
if (!isset($_GET['game'])) {
die("G");
}

include_once("gmolibrary.php");

forceNetread();

$cxn = intDatabase();

$query = "SELECT * FROM games WHERE gameid='" . mysqli_real_escape_string($cxn,
$_GET['game']) . "'";
$result = mysqli_query($cxn,$query);

if ($row = mysqli_fetch_assoc($result)) {
$verifyanswer = round(($_GET['score'] + $row['gameVerifyA']) /
$row['gameVerifyB'],0);
while ($verifyanswer >= $row['gameVerifyC']) {
$verifyanswer = $verifyanswer - $row['gameVerifyC'];
}

if ($_GET['verify'] == $verifyanswer) {
$query = "INSERT INTO highscores (hsGame,hsScore,hsUser) VALUES
(".mysqli_real_escape_string($cxn,
$_GET['game']).",".mysqli_real_escape_string($cxn,
$_GET['score']).",'".mysqli_real_escape_string($cxn,$_GET['user'])."')";
mysqli_query($cxn,$query);
die("1");
} else {
die('0');
}
} else {
die("3");
}
?>
```

Wie man sieht, wird ein Highscore im GET-Feld „score“ nur akzeptiert und in die SQL übernommen, wenn ein „verify“-Feld mit korrektem Inhalt versendet wird. Dazu muss das verify-Feld berechnet werden. Der Algorithmus wurde also aus dem PHP-Code in Arduino-Code übersetzt, so dass „verify“ berechnet werden kann. Die Konstanten VERIFY_A, VERIFY_B und VERIFY_C erhält der Unterhalter der Highscore-Liste bei der Anmeldung.

```
 uint32_t iVerifyAnswer = (uint32_t) ((((float)gameEngine.getPoints() +
VERIFY_A) / VERIFY_B) + 0.5);
```

```
    while(iVerifyAnswer >= VERIFY_C)
      iVerifyAnswer -= VERIFY_C;
```

Die weiteren Felder game und user sind für die von Gamemaker Highscores vergebene Game ID und den frei zu vergebenden Spielernamen gedacht. Allerdings können auch mit Kenntnis aller GET-Felder keine Highscores ohne Weiteres geschrieben werden. Das liegt an einer weiteren Sicherheitssperre von gmhighscores, die die Highscores nicht übernimmt, wenn der in der HTTP-Abfrage enthaltene User-Agent nicht stimmt. So passiert es z.B. das auch bei korrekter Eingabe der URI mit den GET-Feldern als Antwort im Browserfenster „Not allowed!“ erscheint. Das liegt an einer weiteren Abfrage des PHP-Skripts:

```
function
forceNetread() {
$browser = $_SERVER['HTTP_USER_AGENT'];


if ($browser != "netreaddll")
{
die("You are viewing this page with a external web browser. Sorry, but
this is not allowed.");
}
}
```

Die Lösung liegt darin, bei der HTTP-Abfrage, den User-Agent auf „netreaddll“ zu setzen und so dem PHP-Skript vorzugaukeln, man sei die von gmhighscores ausgegebene netread.dll. Dies passiert in der fett markierten Zeile.

```
boolean commitHighscore(uint32_t n_iVerifyAnswer)
{
  boolean success = true;

  success &= sendCmd("AT+CIPSERVER=0","OK");

  success &= configTcpClient();

  success &= sendCmd("AT+CIPSTART=\"TCP\",\"www.gmhighscores.com\",80","OK");

  String sHttpQuery = "GET /newhighscore_action.php?score=";
  sHttpQuery += String(gameEngine.getPoints());
  sHttpQuery += "&user=";
  sHttpQuery += playerName;
  sHttpQuery += "&verify=";
  sHttpQuery += String(n_iVerifyAnswer);
  sHttpQuery += "&game=";
  sHttpQuery += String(GAME_ID);
  sHttpQuery += " HTTP/1.1\r\nHost:www.gmhighscores.com\r\nUser-
Agent:netreaddll\r\n\r\n";

  if(success) {
    success &= sendCmd("AT+CIPSEND=" + String(sHttpQuery.length()), ">");
  }

  if(success) {
    success &= sendCmd(sHttpQuery, "SEND OK");
  }

  if(success) {
    success &= sendCmd("AT+CIPCLOSE", "OK");
  }
```

```
  return success;
}
```

Weitergehende Projekte

Als zukünftige Erweiterungen des Projekts ist der Einsatz optimierter Widerstandswerte geplant. Durch die Vorgaben des Wettbewerbs konnten diese teilweise nicht umgesetzt werden.

Eine Zeitanzeige mit Hilfe einer 7-Segment-Anzeige wäre ebenfalls sehr interessant. So könnte der Spieler genau nachvollziehen, wie viel Zeit ihm noch bleibt.

Zu guter Letzt wäre noch eine Sensorik zu nennen, die den tatsächlichen Durchflug des Quadrocopters durch den Ring detektiert. Momentan ist noch ein Vorbeiflug möglich. Dies ist allerdings mit weiteren Bauteilen verbunden, deren Einsatz für den Kalenderwettbewerb nicht vorgesehen war.

Anhang – Quellcode

Quadrocopter_LandingPad.ino

```
#include <SoftwareSerial.h>
#include <avr/pgmspace.h>
#include "CLandingClearance.h"
#include "CGameEngine.h"
#include "rgb.h"
#include "espUtil.h"

#define DEBUG

#define VERIFY_A  5809603.f
#define VERIFY_B       67.f
#define VERIFY_C    47060
#define GAME_ID      4736

#define LED_WLAN         13
                            // Pin 6 = AIN0 (+)
                            // Pin 7 = AIN1 (-)
#define PIN_ANGLE_SENSOR  2  // Pin 2 = Neigungssensor
#define PIN_POS_LIGHT     3  // Pin 3 = Pos. Light
#define PIN_START_SWITCH 10  // Pin 10= Start Switch

const char site[] PROGMEM = {"<html><head><title>Quadrocopter Landing
Site</title></head><body style=\"background-color: black; color: green; font-
weight: bold\"><h2 style=\"color: red; box-shadow: 3px 3px 3px 2px #444 inset;
text-align: center\">Quadrocopter Challenge</h2>Aktuelle Highscores:<br><iframe
src =\"http://www.gmhighscores.com/highscores_iframe.php?gameid=4736\"
width=\"400\" height=\"300\"><p>Your browser does not support
iframes.</p></iframe><br><p>Geben Sie den Spielernamen ein:<form
method=\"get\"><input type=\"text\" name=\"playerName\" value=\"#playername#\"
style=\"font-size: 14pt; color: blue\"><input type=\"submit\" value=\"Submit\"
style=\"font-size: 14pt\"></form></p></body></html>"};
// #playername#


CLandingClearance landingClearance;
CGameEngine       gameEngine;
String            playerName;

// ================================================
// void setup() : Startup actions. Press and hold the START button to bring the
//                board into a config Mode, where the barrier for the Analog
Comparator
//                for the landing sensor can be set.
// ================================================
void setup() {
  playerName = "unnamed";

  setupAnalogComparator();

  Serial.begin(19200);
  esp8266.begin(19200);

  esp8266.println("AT+RST");

  delay(5000);      // Give the ESP8266 enough time to initialize.

  pinMode(PIN_START_SWITCH, INPUT_PULLUP);
```

```
  pinMode(LED_WLAN, OUTPUT);
  pinMode(PIN_ANGLE_SENSOR, INPUT);
  pinMode(PIN_POS_LIGHT, OUTPUT);
  DDRC &= ~((1<<PINC0) | (1<<PINC1) | (1<<PINC2) | (1<<PINC3) | (1<<PINC4) |
(1<<PINC5));   // All Pins are input pins (we want to use the internal pullup
resistors)

  // Try all colors of the RBG LED to give the user a visual indication of the
initialization
  switchOnBlue();
  delay(200);
  switchOffBlue();
  delay(200);
  switchOnRed();
  delay(200);
  switchOffRed();
  delay(200);
  switchOnGreen();
  delay(200);
  switchOffGreen();
  delay(200);

  digitalWrite(LED_WLAN, LOW);
  analogWrite(PIN_POS_LIGHT, 0xff);

  if(!digitalRead(PIN_START_SWITCH))            // If the switch is pressed,
enter the configuration mode
  {
    switchOffBlue();
    switchOnRed();
    switchOnGreen();
    // If start switch is pressed
    while(!digitalRead(PIN_START_SWITCH))       // Run while switch is
depressed, so that the Poti can be adjusted, according to actual light
conditions
    {
      if(isLanded())
      {
        switchOnBlue();
      }
      else
      {
        switchOffBlue();
      }
    }
    switchOffBlue();
    switchOffRed();
    switchOffGreen();
  }

#ifdef DEBUG
  Serial.println("Initializing...");
#endif

  boolean bInitSuccessful = true;

  bInitSuccessful &= espConfig();
  if(bInitSuccessful)
  {
    do {

#ifdef DEBUG
    Serial.println("Trying to join AP:" + SSID);
#endif
```

```
    bInitSuccessful &= configStation();

#ifdef DEBUG
    if(!bInitSuccessful) Serial.println("Could not join AP!");
#endif

    } while(!bInitSuccessful);
    bInitSuccessful = true;
  }
  if(bInitSuccessful)
  {
#ifdef DEBUG
    Serial.print("Initialized! WLAN joined! IP:");
    Serial.println( sendCmd("AT+CIPSTA?") );
    Serial.print("Trying to ping 8.8.8.8...");
#endif
    bInitSuccessful &= sendCmd("AT+PING=\"8.8.8.8\"", "OK");        // Try to
ping google...just to be sure, that there is a connection to the internet
    bInitSuccessful &= configTcpServer();
#ifdef DEBUG
    if(bInitSuccessful)
      Serial.println("successful!");
    else
      Serial.println("failed!");
#endif
  }
  else
  {
    Serial.println("Initialization failure!");
  }

  if(bInitSuccessful)
  {
    digitalWrite(LED_WLAN, HIGH);
  }
  else {
    serialDebug();
  }


}


// ================================================
// void setupAnalogComparator(): Sets the Analog Comparator up.
// ================================================
void setupAnalogComparator()
{
  ACSR &= ~(1<<ACD);        // Clear ACD bit to switch Analog Comparator on
  ACSR &= ~(1<<ACBG);       // Switch off bandgap reference
  ACSR &= ~(1<<ACIE);       // Clear ACIE bit to switch off Interrupt
  ADCSRB &= ~(1<<ACME);     // Clear ACME bit to connect AIN1 to the negative
input of the Analog Comparator
  DIDR1 |= ((1<<AIN1D) | (1<<AIN0D));
                            // Disable the Digital Input Buffers
}


boolean LastLanded = 0;
boolean LastAngleSensor = false;
boolean LastStartSwitch = true;
```

```
// ====================================================
// void loop(): This function is being run continuously, after the
initialization was successfully completed.
// ====================================================
void loop() {

  uint8_t BrightnessPosLights = ((millis() >> 2) % 256);     // Range 0...255,
calculate a dim value, depending on the time

  // If the game is actually running, let the "Position Lights" be illuminated
and dimmed up and down continuously!
  if(gameEngine.isRoundRunning()) {

    if(BrightnessPosLights > 127)                           // If the value is >
127, start to dim the lights down
    {
      BrightnessPosLights = 255 - BrightnessPosLights;
    }

    BrightnessPosLights <<= 1;

    analogWrite(PIN_POS_LIGHT, BrightnessPosLights);    // Dim the lights up and
down
  }
  else {
    analogWrite(PIN_POS_LIGHT, 0);                      // If the round is not
running, switch the landing strip lights off

    if(LastStartSwitch != digitalRead(PIN_START_SWITCH))  // If the switch state
has been changed
    {
      if(!digitalRead(PIN_START_SWITCH))                // switch is pressed
      {
        gameEngine.startRound();
      }
    }
  }
  LastStartSwitch = digitalRead(PIN_START_SWITCH);


  if( digitalRead(PIN_ANGLE_SENSOR) != LastAngleSensor )  // Trigger on the edge
  {
    if(!digitalRead(PIN_ANGLE_SENSOR))                    // Low is active,
since there is a BC547 used as an inverter!
    {
    // if the stargate has been touched
      gameEngine.decreasePoints(100);
    }
  }
  LastAngleSensor = digitalRead(PIN_ANGLE_SENSOR);        // Store the last
Value, so that we can trigger on the edge

  if(gameEngine.isRoundRunning()) {
    if(landingClearance.isControlled())
    {
      if(landingClearance.isCleared()) {
        switchOffBlue();
        switchOffRed();
        switchOnGreen();
      }
      else {
        switchOffBlue();
        switchOnRed();
        switchOffGreen();
```

```
      }
    }
    else {
       switchOnBlue();
       switchOffRed();
       switchOffGreen();
    }
  }

  if(gameEngine.isRoundRunning()) {
    if(LastLanded != isLanded())
    {
      if(isLanded())
      {
        if(landingClearance.isControlled())
        {
          if(landingClearance.isCleared())
          {
            gameEngine.increasePoints(100);
          }
          else
          {
            gameEngine.decreasePoints(100);
          }
        }
      }
      else
      {
          // After takeoff!!!
          landingClearance.afterTakeoff();
      }
    }
  }
  LastLanded = isLanded();



  if( gameEngine.checkRoundFinished() )
  {

    // Round is finished, submit points
    uint32_t iVerifyAnswer = (uint32_t) ((((float)gameEngine.getPoints() +
VERIFY_A) / VERIFY_B) + 0.5);
    while(iVerifyAnswer >= VERIFY_C)
      iVerifyAnswer -= VERIFY_C;

     if( commitHighscore(iVerifyAnswer) )
        configTcpServer();
#ifdef DEBUG
  Serial.println("Points:");
  Serial.println(gameEngine.getPoints(), DEC);
#endif
  }


  // =================================================
  // Webserver part
  // =================================================
  if(esp8266.available())
  {
    // Data is available
    if(esp8266.find("+IPD,"))
    {
      int iConnectId = esp8266.parseInt();
```

```
      if(esp8266.findUntil("?playerName=", "\n"))
      {

        playerName = esp8266.readStringUntil(' ');
        playerName.replace("+"," ");
#ifdef DEBUG
        Serial.print("?playerName : ");
        Serial.println(playerName);
#endif
      }

      if(sendWebsite(iConnectId) )
      {
#ifdef DEBUG
        Serial.println("Website sent! OK!");
#endif
      }
    }

  }
}


// ================================================
// boolean isLanded(): Checks the value of the Analog Comparator in order to
check, whether the Quadrocopter has landed.
//
// ================================================
// true:  Quadrocopter landed
// false: Quadrocopter not landed
// ================================================
boolean isLanded() {
  boolean ReturnValue = false;
  if(ACSR & (1<<ACO)) {
    // AIN0 > AIN1
#ifdef DEBUG
    //Serial.println("AIN0 > AIN1");
#endif
    ReturnValue = false;
  }
  else {
#ifdef DEBUG
    //Serial.println("AIN0 < AIN1");
#endif
    ReturnValue = true;
  }
  return ReturnValue;
}


// ================================================
// boolean sendWebsite(int iConnectionId): Prepare the website and send it to
the client.
// ================================================
// int iConnectionId: Connection Identifier, required since in the Server Mode
we have CIPMUX=1, so multiple
//                    connections are possible, which have to be identified.
// ================================================
// true: Website sent successfully.
// false:An error occured.
// ================================================
boolean sendWebsite(int iConnectionId)
{
  boolean success = true;
```

```
  String webpage;

  char tempChar;
  for(int i = 0; i < sizeof(site); i++)
  {
    tempChar = pgm_read_byte_near(site + i);
    webpage += tempChar;
  }


  webpage.replace("#playername#", playerName);

#ifdef DEBUG
  Serial.println("Site:");
  Serial.print(webpage);
#endif

  if(sendCmd("AT+CIPSEND=" + String(iConnectionId) + "," +
String(webpage.length()+2), ">"))
  {
    esp8266.println(webpage);
    esp8266.find("SEND OK");

    success &= true;
    success &= sendCmd("AT+CIPCLOSE=" + String(iConnectionId), "OK");
  }
  else
  {
    Serial.println("Error CIPSEND");
    success = false;
    sendCmd("AT+CIPCLOSE=" + String(iConnectionId));
  }
  return success;


}

// ================================================
// boolean commitHighscore(uint32_t n_iVerifyAnswer): Commits the score of the
actual round to the gamemaker Highsore service.
// ================================================
// uint32_t n_iVerifyAnswer: Required for the GET-Query.
// ================================================
// true: Highscore successfully sent.
// false:An error occured.
// ================================================
boolean commitHighscore(uint32_t n_iVerifyAnswer)
{
  boolean success = true;

  success &= sendCmd("AT+CIPSERVER=0","OK");

  success &= configTcpClient();

  success &= sendCmd("AT+CIPSTART=\"TCP\",\"www.gmhighscores.com\",80","OK");

  String sHttpQuery = "GET /newhighscore_action.php?score=";
  sHttpQuery += String(gameEngine.getPoints());
  sHttpQuery += "&user=";
  sHttpQuery += playerName;
  sHttpQuery += "&verify=";
  sHttpQuery += String(n_iVerifyAnswer);
  sHttpQuery += "&game=";
  sHttpQuery += String(GAME_ID);
```

```
  sHttpQuery += " HTTP/1.1\r\nHost:www.gmhighscores.com\r\nUser-
Agent:netreaddll\r\n\r\n";

  if(success) {
    success &= sendCmd("AT+CIPSEND=" + String(sHttpQuery.length()), ">");
  }

  if(success) {
    success &= sendCmd(sHttpQuery, "SEND OK");
  }

  if(success) {
    success &= sendCmd("AT+CIPCLOSE", "OK");
  }

  return success;
}
```

CGameEngine.h

```
#ifndef CGAMEENGINE
#define CGAMEENGINE

// ====================================================
// CGameEngine: Class. Defining the Game Engine, i.e. the behaviour of the game.
// ====================================================
class CGameEngine {
  private:
    int16_t  m_iPoints;
    uint32_t  m_iRoundStartTime;
    boolean   m_bRoundRunning;
    uint8_t   m_Phase;
  public:
    const uint8_t PHASE_RED = 0;
    const uint8_t PHASE_GREEN = 1;
    const uint8_t PHASE_BLUE = 2;

// ====================================================
// CGameEngine(): Constructor
// ====================================================
    CGameEngine() {
      m_iPoints = 0;
      m_iRoundStartTime = 0;
      m_bRoundRunning = false;
      m_Phase = PHASE_RED;
    }


// ====================================================
// CGameEngine::CGameEngine(int n_iPoints):
// Constructor. Sets the point count.
// ====================================================
    CGameEngine(int n_iPoints) {
      m_iPoints = n_iPoints;
      m_iRoundStartTime = 0;
      m_bRoundRunning = false;
      m_Phase = PHASE_RED;
    }


// ====================================================
// void CGameEngine::increasePoints(int iPoints):
// Increases the point count by the attrubute value.
// ====================================================
    void increasePoints(int iPoints) {
      m_iPoints += iPoints;
#ifdef DEBUG
      Serial.println(m_iPoints, DEC);
#endif
    }


// ====================================================
// void CGameEngine::decreasePoints(int iPoints):
// Decreases the point count by the attribute value.
// ====================================================
    void decreasePoints(int iPoints) {
      m_iPoints -= iPoints;
      if(m_iPoints < 0)
          m_iPoints = 0;
#ifdef DEBUG
```

```
      Serial.println(m_iPoints, DEC);
#endif
    }


// ================================================
// uint16_t CGameEngine::getPoints:
// Returns the point count.
// ================================================
    uint16_t getPoints() {
      return m_iPoints;
    }


// ================================================
// void CGameEngine::setPoints(uint16_t n_iPoints):
// Gives the possibility to manipulate the point count.
// ================================================
    void setPoints(uint16_t n_iPoints) {
      m_iPoints = n_iPoints;
    }


// ================================================
// void CGameEngine::startRound():
// Starts a new round. Sets the required members.
// ================================================
    void startRound() {
        m_iPoints = 0;
        m_iRoundStartTime = millis();
        m_bRoundRunning = true;
        m_Phase = PHASE_BLUE;
    }


// ================================================
// boolean CGameEngine::checkRoundFinished():
// Checks whether the round is finished (time), and sets the members if
necessary.
// ================================================
    boolean checkRoundFinished() {
      boolean RoundFinished = false;
      if(m_bRoundRunning) {
        uint32_t iTimeElapsed = millis() - m_iRoundStartTime;
        if(iTimeElapsed > 60000) {
          m_bRoundRunning = false;
          RoundFinished = true;
        }

      }
      return RoundFinished;
    }


// ================================================
// boolean CGameEngine::isRoundRunning()
// Returns m_bRoundRunning.
// ================================================
    boolean isRoundRunning() {
      return m_bRoundRunning;
    }

};
#endif
```

CLandingClearance.h

```
#ifndef CLANDINGCLEARANCE
#define CLANDINGCLEARANCE


// ================================================
// CLandingClearance: Class. Defines whether we have a landing clearance or not.
// ================================================
class CLandingClearance {

  private:
  boolean m_bClearedToLand;
  boolean m_bControlled;
  uint32_t m_iDepartureTime;
  uint32_t m_iTimeInAir;

  public:
  CLandingClearance() {
    m_bClearedToLand = false;
    m_bControlled = true;
    m_iDepartureTime = millis();
    m_iTimeInAir = 0;
  }

  ~CLandingClearance() {

  }

  boolean isCleared() {
    return m_bClearedToLand;
  }

  boolean isControlled() {
    uint32_t iTimeInAir = millis() - m_iDepartureTime;
    if(iTimeInAir < 5000) {       // If time in air < 5s, i.e. give the pilot 5
s to depart his quadrocopter
      m_bControlled = false;
      m_bClearedToLand = false;
    }
    else
    {
      m_bControlled = true;
      if(iTimeInAir != m_iTimeInAir)
      {
        if(!(iTimeInAir % 8000))      // If iTimeInAir has changed and can be
divided by 8000: Toggle clearance state
        {
          m_bClearedToLand = !m_bClearedToLand;

        }
      }
    }

    m_iTimeInAir = iTimeInAir;

    return m_bControlled;
  }

  void ClearedToLand() {
    m_bClearedToLand = true;
  }

  void CancelLandingClearance() {
```

```
    m_bClearedToLand = false;
  }

  void setControlled(boolean n_bControlled) {
    m_bControlled = n_bControlled;
    if(!m_bControlled) {
      m_bClearedToLand = false;
    }
  }

  void afterTakeoff() {
    m_iDepartureTime = millis();
  }


};

#endif
```

espUtil.h

```
#ifndef ESPUTIL_H
#define ESPUTIL_H

// ================================================
// espUtil.h: Utility functions used for the esp8266 communication, such as the
AT-Commands, etc.
// ================================================

#define SSID String("[Your SSID]")
#define PWD  String("[Your Password]")

SoftwareSerial esp8266(11,12);    // RX, TX

boolean sendCmd(String cmd, char* expectedAnswer)
{
  esp8266.println(cmd);
  if(esp8266.findUntil(expectedAnswer, "ERROR"))
  {
    return true;
  }
  else
  {
    Serial.println("ESP SEND ERROR: " + cmd);
    return false;
  }
}

String sendCmd(String cmd)
{
  esp8266.println(cmd);
  return esp8266.readString();
}

void serialDebug()
{
  while(true)
  {
    if(esp8266.available())
      Serial.write(esp8266.read());
    if(Serial.available())
      esp8266.write(Serial.read());
  }
}

boolean espConfig()
{
  boolean success = true;
  esp8266.setTimeout(20000);
  success &= sendCmd("AT", "OK");
  esp8266.setTimeout(1000);


  success &= sendCmd("AT+CWMODE=1", "OK");



  return success;
}

boolean configStation()
{
```

```
  boolean success = true;
  esp8266.setTimeout(15000);
  success &= sendCmd("AT+CWDHCP=1,1", "OK");
  success &= sendCmd("AT+CWJAP=\"" + SSID + "\",\"" + PWD + "\"","OK");
  esp8266.setTimeout(1500);
  //success &= sendCmd("AT+CIPMODE=0", "OK");
  success &= sendCmd("AT+CIPMUX=1", "OK");
  return success;

}

boolean configTcpServer() {
  boolean success = true;
  success &= sendCmd("AT+CIPMUX=1","OK");
  success &= sendCmd("AT+CIPSERVER=1,80","OK");

  return success;
}

boolean configTcpClient() {
  boolean success = true;
  success &= sendCmd("AT+CIPMUX=0","OK");

  return success;
}




#endif
```

rgb.h

```
#ifndef RGB_H
#define RGB_H

// ================================================
// rgb.h: Utillity functions, used to switch single RGB-LED colors on or off.
// We use a parallel connection of two neighboring pins, so that we have a total
output impedance of 1/2*20KR.
// Therefore it is crucial to switch two Ports On/Off simulataneously!!
Otherwise there will be a current flow
// between the pins!
// ================================================

void switchOnBlue()
{
  PORTC |= ((1<<PINC2) | (1<<PINC3));
}

void switchOffBlue()
{
  PORTC &= ~((1<<PINC2) | (1<<PINC3));
}

void switchOnRed()
{
  PORTC |= ((1<<PINC4) | (1<<PINC5));
}

void switchOffRed()
{
  PORTC &= ~((1<<PINC4) | (1<<PINC5));
}

void switchOnGreen()
{
    PORTC |= ((1<<PINC0) | (1<<PINC1));
}

void switchOffGreen()
{
  PORTC &= ~((1<<PINC0) | (1<<PINC1));
}

#endif
```