Alarmanlage mit Arduino Nano und dem Lage- und Beschleunigungssensor GY-521

Die ursprüngliche Idee war der Bau eines kleinen Spielzeugs für eine gesellschaftliche Unterhaltung. Ein Freund suchte zum spielen für eine gesellige Trinkrunde ein auf Bewegung empfindliches Gerät. Als erstes Denkt man dann an das Geschicklichkeitsspiel "Der heiße Draht". Hier wird versucht, in möglichst kurzer Zeit eine kleine Drahtschlaufe durch einen selbst entworfenen Parcours, der aus einem gebogenen Draht besteht, zu durchfahren, ohne diesen zu berühren. Ist leider etwas zu gewöhnlich, vielleicht ist etwas mit der Abhängigkeit der Lage und die Erfassung der Beschleunigung mal was interessanter. Jedenfalls wurde das weitere Gespräch über 3-Achsen Gyroskope und 3-Achsen Beschleunigungssensoren, womit 6 Freiheitsgrade gleichzeitig erkennbar sein sollen, recht unterhaltsam.
Jetzt weiß ich was ein Gyroscope und ein Accelerometer ist, was mich aber nicht davon abhält, diese immer mal wieder zu verwechseln. Wer schon einmal ein neuartiges und innovatives Fortbewegungsmittel in Verbindung mit enormem Fahrspaß und dem Gefühl der Grenzenlosigkeit gefahren hat, kennt die sich selbst ausgleichenden Fahrroboter wie Segway und Ninebot. Diese funktionieren Hauptsächlich durch den Einsatz eines elektronischen Gyroscope in Kombination mit einem elektronischen Accelerometer

Da mir das alles zu komplex und kompliziert wurde und ich nicht zum Gespräch beitragen konnte, suchte ich mir ein eigenes Gedankenspiel nach dem Motto, die Gedanken sind frei.

In meinem virtuellen Tagebuch habe ich auch einiges darüber notiert:

Eine kleine Schaltung aus zusammengefügten Arduino Nano mit dem Sensor GY-521 soll demonstrieren, mit welch einfachen Mitteln heutzutage eine einfache Alarmanlage realisiert werden kann. Dieses Beispiel ist keine Endlösung und soll vom Anwender gemäß seiner eigenen Vorstellung von ihm angepasst werden.

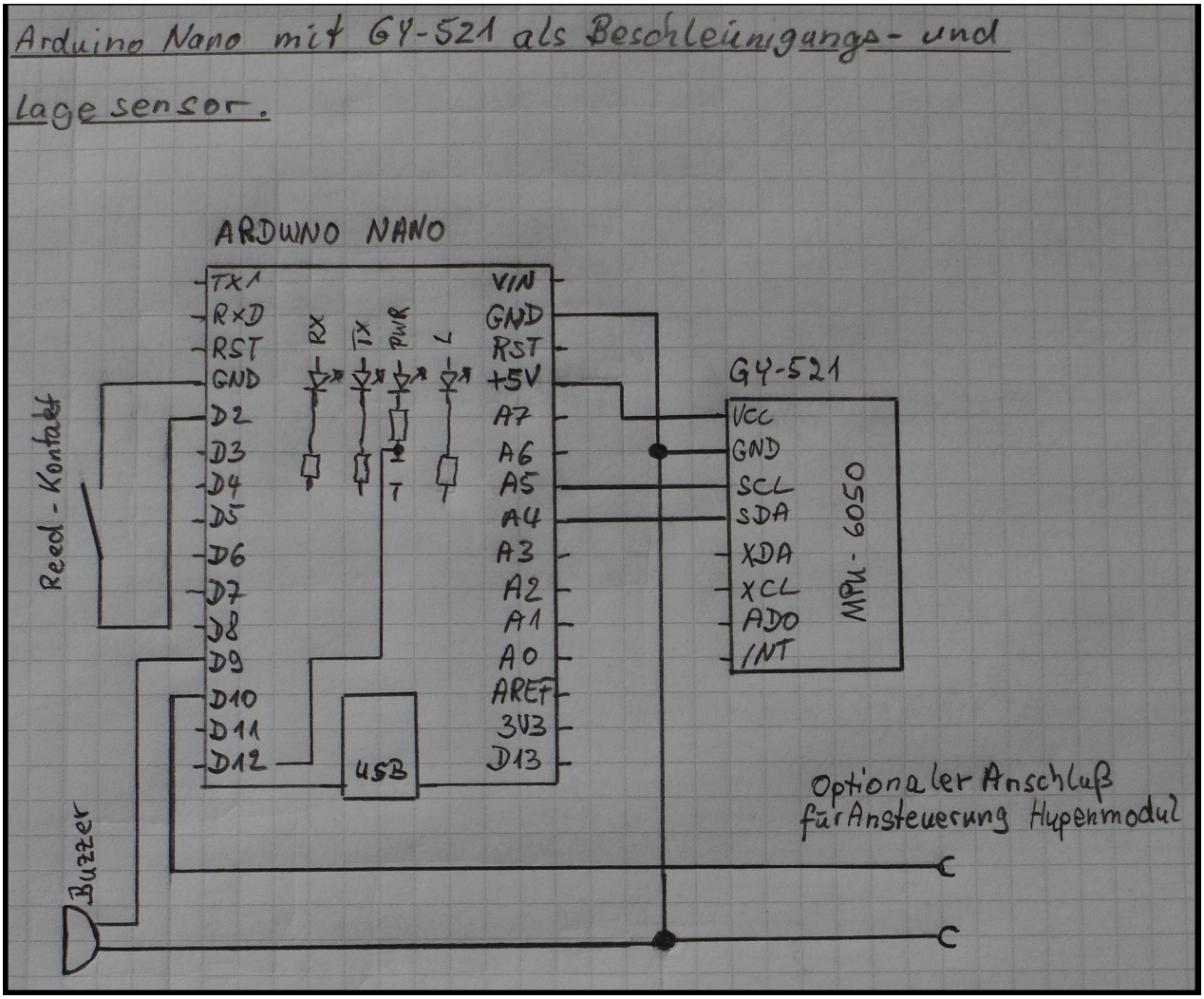
Die Anlage wird mit einem Reedkontakt aktiviert und deaktiviert. Mittels eines Summers werden die Zustände der Funktionen und ein Alarmzustand signalisiert. Ein anstehender Alarm wird nach einiger Zeit zurückgenommen und ist bei jeder neuen Lageänderung erneut für eine kurze Zeitdauer aktiv. Die Anlage kann jederzeit über den Reedkontakt, auch im Alarmzustand, deaktiviert werden.

Mit zwei Leuchtdioden wird der Zustand signalisiert.
Daraus ergeben sich die Zustände „Anlage Aus“ (LED1, Rot ein; LED2, Grün aus), „Anlage Ein“ (LED1, Rot aus; LED2, Grün ein) und „Alarmauslösung“ (LED1, Rot ein; LED2, Grün ein).
Zusätzlich werden diese Zustände über dem Summer signalisiert. Dreimal Piepsen heißt Anlage ist aktiviert worden. Nach einer kurzen Verweildauer erfolgt ein einmaliges Piepsen und bedeutet Alarmüberwachung aktiviert. Das Deaktivieren der Anlage wird mit einem zweimaligen Piepsen quittiert.
Eine ausreichende Erschütterung, Beschleunigung oder Bewegung führt bei aktiver Alarmüberwachung für eine gewisse Zeitdauer zu einem intermittierenden Piepsen und wiederholt sich bis eine neue Ruhelage festgestellt wird.

Die Empfindlichkeit kann in zwei Stufen ausgewählt werden und ist in der Software definiert. Wird der Reedkontakt beim Anlegen der Versorgungsspannung oder bei Auslösung eines Resets betätigt, so wird ein alternativer Datensatz anstelle des normalen Datensatz geladen.

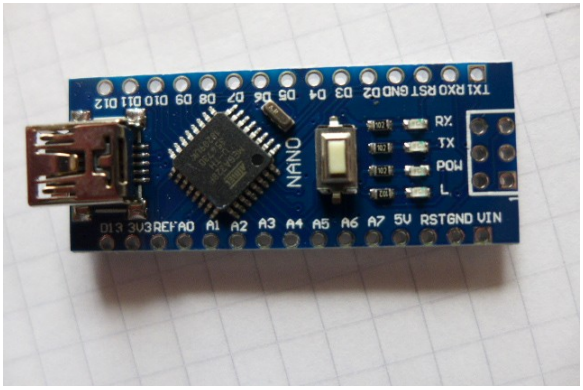
Für den Zusammenbau der Schaltung benötigen wir einen Arduino Nano, einen Gyroscope mit Accelerometer GY-521, einen passiven Summer und einen Reedkontakt mit ensprechenden Magneten für dessen Betätigung.
Für den späteren Betrieb kann eine Power-Bank als Spannungsversorgung dienen. Hier ist darauf zu achten, dass sich diese bei zu geringer Belastung nicht selbst deaktiviert. Die für die Programmierung des Arduino benötigte Programmierleitung wird später im Dauerbetrieb als Versorgungsleitung zwischen Arduino und der Power-Bank verwendet.

Nachdem wir uns mit nötigen Teilen versorgt haben, können wir diese jetzt nach dem folgenden Schaltbild verbauen.

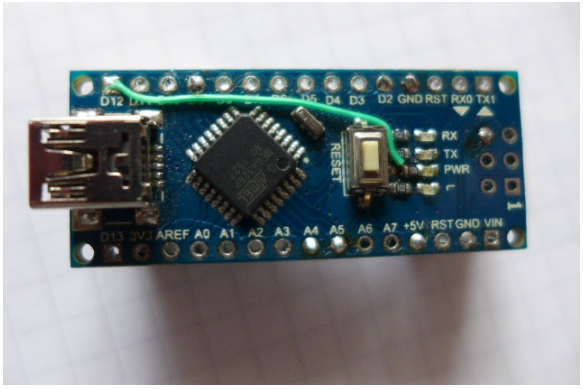


Zwei Dinge fallen sofort ins Auge, erstens ist ein Ausgang für ein Hupenmodul vorgesehen und es gibt keine zusätzlichen Leuchtdioden.
Der Ausgang (D10, Pin 10) des Hupenmodul ist nur zur Ansteuerung von Hupen mit einem eigenen Leistungsteil geeignet. Die Last der Ansteuerungsleitung darf 20 mA nicht überschreiten, was über eine Treiberstufe gewährleistet werden muss. Dieser Ausgang wird in der Software angewendet und ist bei Alarmauslösung aktiv. Durch diese optionale nicht verdrahtete Funktion ist es möglich, diese Schaltung zum Beispiel als Fahrradalarmanlage einzusetzen. Meine Oma findet einen anderen Verwendungszweck viel sinnvoller. Sie meint, sie fühle sich sicherer im Cafe und auf dem Bahnhof, wenn dies olle Ding in ihrer Handtasche neben ihr zusammen rumliegen tut. Jetzt brauchen wir bald einen „zu gossen Abstands-Alarm“, denn Oma ist manchmal auch etwas tüdelig und nimmt nicht immer alles wieder mit nach Hause.

Um auf die Leuchtdioden zurück zu kommen, Tüdeligkeit spielt dabei keine Rolle. Da nun mal wieder etwas sinnlos rumhängen könnte, was diesmal bewusst vermieden werden sollte, wird eine Leuchtdiode auf dem Arduino Nano umverdrahtet. Die erste Leuchtdiode ist eine rote, diese wird über D13 angesteuert und ist standardmäßig verdrahtet. Die zweite Leuchtdiode ist grün und ist standardmäßig die Power-Anzeige. Für den Umbau wird die Verbindung zwischen der grünen Leuchtdiode und dem Vorwiderstand aufgetrennt. Danach wird der Vorwiderstand entnommen und auf die aufgetrennte Verbindungsstelle direkt an die Leuchtdiode gelötet. Das freie Ende des Vorwiderstand wird mittels einer Drahtbrücke mit dem Punkt D12 (Pin 12) verbunden.

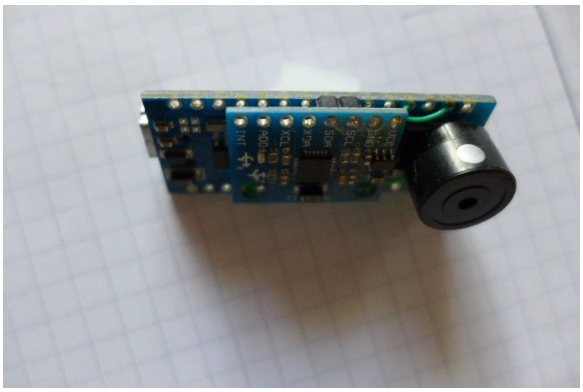


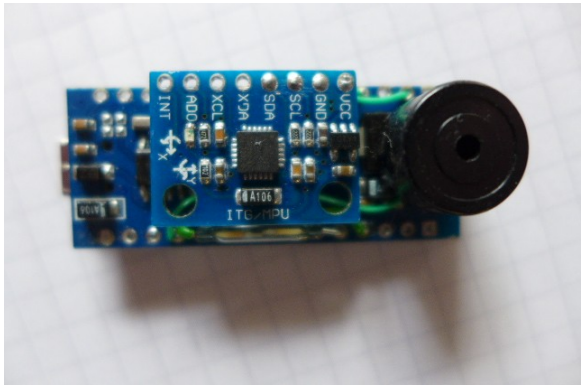
Vor dem Umbau der Leuchtdiode



Nach dem Umbau der Leuchtdiode

Der weitere Aufbau geht jetzt rascher vonstatten. Die Platine mit Gyroscope und Accelerometer GY-521 wird auf den Arduino Nano aufgelötet und der Reedkontakt wird zwischen den beiden Platinen platziert. Mit etwas Geschick passt der Buzzer auch noch auf dem Arduino Nano.


Anbindung des GY-521 auf dem Arduino Nano

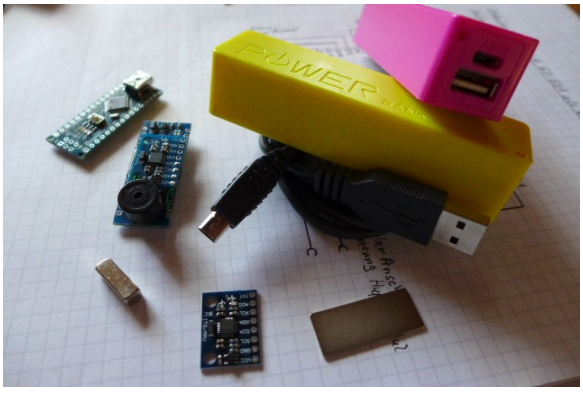


Positionen von Reedkontakt und Buzzer

Der Aufbau der Schaltung ist jetzt abgeschlossen und es kann eine Spannungsversorgung angeschlossen werden. Die Leuchtdiode auf dem GY-521 dient als Kontrollanzeige „Versorgungsspannung ist angelegt“ und kann die ursprüngliche Funktion der umgebauten Leuchtdiode übernehmen.

Im rechten Bild ist der Typ einer Power-Bank zu sehen, die für besonders geringe Ströme geeignet ist und schon mit Strömen um 2 mA dauerhaft in den Arbeitsbereich (5 V) übergeht.

Im weiteren Teil wird auf die Programmierung des Arduino Nano eingegangen und einige Funktionen kurz erläutert.


Empfohlene Power-Bank, GY-521, Arduino Nano etc.

Das Programm des Arduino Nano mit GY-521

Um den Arduino Nano programmieren zu können, muss zunächst die Arduino Software (IDE) auf dem Rechner installiert sein. Nach dem Start der Arduino IDE wird die vorher auf dem Rechner hinterlegte Datei „SFB_MPU6050_Alarmanlage_Version_Z12“ geöffnet und danach über die USB-Verbindung zum Arduino Nano übertragen. Als Dokumentation ist ein Ausdruck des Programm empfehlenswert.

Der Kommentarblock beschreibt kurz die Programmfunktion

Jedes Programm sollte eine kurze Beschreibung der Funktion des Programm und die Angabe des Ausgabestandes enthalten.

```
//  ********************************************************************************************
//
//  Alarmanlage mit Lageerfassung und Beschleunigungserkennung
//
//  Normale Empfindlichkeit bei nicht Betätigung des Tasters "Alarmanlage Ein" bei Neustart.
//  Alternative Empfindlichkeit bei Betätigung des Tasters "Alarmanlage Ein" bei Neustart.
//
//  Bei Alarmauslösung wird danach die neue Lage ermittelt. Jede weitere Lageänderung führt
//  zur Alarmauslösung.
//
//
//  (c) Jan von Kölle 14.07.2018 V0.1
//
//  ********************************************************************************************
```

Einbindung der benötigten Bibliotheken

Als erstes werden die benötigten Bibliotheken ins Programm eingebunden. Da der GY-521 über die I2C-Schnittstelle kommuniziert wird die Standard-Bibliothek „Wire.h“ benötigt.

```
//  ********************************************************************************************

// Import Standard Library
#include<Wire.h>                          // Software I2C

//  ********************************************************************************************
```

Der Block „#define“

Danach werden die im Programm verwendeten Bedien- und Ausgabeelemente definiert und den Pins des Arduino Nano zugeordnet.

```
//  ********************************************************************************************

#define       Taster          2           // Taster
#define       LedRot         13           // LED Rot
#define       LedGruen       12           // LED Gruen
#define       Summer          9           // Summer
#define       Hupe           10           // Intervall Piezo (Hupenmodul)

//  ********************************************************************************************
```

Vorgabewerte normale und alternative Empfindlichkeit ändern

Die Vorgabewerte für normale und alternative Empfindlichkeit der Alarmauswertung (Funktion Alarmauslösung) stehen am Anfang des Programm und kann angepasst werden. Für jede aufgeführte Achse können unterschiedliche Werte eingetragen werden. In diesem Beispiel ist für die normale Empfindlichkeit der Wert „20.“ für jede Achse eingetragen.

```
//  ********************************************************************************************

//  Hier können die Vorgabewerte für normale und alternative Empfindlichkeit der
//  Alarmauswertung (Funktion Alarmauslösung) angepasst werden.

#define        Normal_ACCEL_GradX    10.     // Normale Empfindlichkeit Accel X-Achse
#define        Normal_ACCEL_GradY    10.     // Normale Empfindlichkeit Accel Y-Achse
#define        Normal_GYRO_BesX      15.     // Normale Empfindlichkeit Gyro X-Achse
#define        Normal_GYRO_BesY      15.     // Normale Empfindlichkeit Gyro Y-Achse
#define        Normal_GYRO_BesZ      30.     // Normale Empfindlichkeit Gyro Z-Achse

#define        Altern_ACCEL_GradX    20.     // Alternative Empfindl. Accel X-Achse
#define        Altern_ACCEL_GradY    20.     // Alternative Empfindl. Accel Y-Achse
#define        Altern_GYRO_BesX      30.     // Alternative Empfindl. Gyro X-Achse
#define        Altern_GYRO_BesY      30.     // Alternative Empfindl. Gyro Y-Achse
#define        Altern_GYRO_BesZ      60.     // Alternative Empfindl. Gyro Z-Achse

//  ********************************************************************************************
```

Die globalen Variablen

Hier werden die globalen Variablen definiert und eventuell mit Vorgabewerten besetzt.

```
//  ********************************************************************************************

const int      MPU_addr=0x68;                        // I2C Addresse MPU-6050

int16_t        GYRO_OutX,  GYRO_OutY,  GYRO_OutZ;   // Gyroscope (Winkeldrehung)
int16_t        TEMP_Out;                             // Temperatur
int16_t        ACCEL_OutX, ACCEL_OutY, ACCEL_OutZ;  // Accelerometer (Beschleunigung)

float          GYRO_MitX,  GYRO_MitY,  GYRO_MitZ;   // Mittelwert Gyroscope (Winkeldrehung)
float          TEMP_Mit;                             // Mittelwert Temperatur
float          ACCEL_MitX, ACCEL_MitY, ACCEL_MitZ;  // Mittelwert Accelerometer (Beschleunigung)

double         Alt_ACCEL_GradX, ACCEL_GradX;         // Accelerometer X-Lage in Grad
double         Alt_ACCEL_GradY, ACCEL_GradY;         // Accelerometer Y-Lage in Grad
double         Alt_ACCEL_GradZ, ACCEL_GradZ;         // Accelerometer Z-Lage in Grad

double         Alt_TEMP_Grad, TEMP_Grad;             // Temperatur in Grad

double         Alt_GYRO_BesX, GYRO_BesX;             // Gyroscope X-Lage Beschleunigung
double         Alt_GYRO_BesY, GYRO_BesY;             // Gyroscope Y-Lage Beschleunigung
double         Alt_GYRO_BesZ, GYRO_BesZ;             // Gyroscope Z-Lage Beschleunigung

byte           Alarmanlage           = 0;            // Alarmanlage aus/ein
byte           AlarmanlageScharf     = 0;            // Alarmanlage scharf aus/ein
byte           Alarm                 = 0;            // Alarmauslösung aus/ein

long           StartZeitpunkt;                       // Zeitmessung Startzeit

double         Grenzw_ACCEL_GradX;                   // Grenzwert Empfindl Accel X-Achse
double         Grenzw_ACCEL_GradY;                   // Grenzwert Empfindl. Accel Y-Achse
double         Grenzw_GYRO_BesX;                     // Grenzwert Empfindl. Gyro X-Achse
double         Grenzw_GYRO_BesY;                     // Grenzwert Empfindl. Gyro Y-Achse
double         Grenzw_GYRO_BesZ;                     // Grenzwert Empfindl. Gyro Z-Achse

//  ********************************************************************************************
```

Die „void setup()“ Routine

Die Routine „void setup()“ initialisiert den Arduino Nano und wird nur einmal beim Start des Programm durchlaufen.
Es werden die Ein- und Ausgänge initialisiert und die Werte für die Normale- bzw. Alternative Empfindlichkeit in Abhängigkeit des Reedkontakt geladen. Zum Schluss wird die I2C-Schnittstelle aktiviert und die MPU6050 des GY-521 in Betrieb genommen. Wer sich für die Funktion und die Register der MPU6050 interessiert kann die relevanten Informationen aus dem Datenblatt der MPU6050 entnehmen.

```
//  ********************************************************************************************

  void setup()
    {
      pinMode(Taster,    INPUT);                     // Taster

      digitalWrite(Taster, HIGH);                    // LED Rot ein

      pinMode(Summer,   OUTPUT);                     // Summer
      pinMode(Hupe,     OUTPUT);                     // Intervall Piezo
      pinMode(LedRot,   OUTPUT);                     // LED Rot
      pinMode(LedGruen, OUTPUT);                     // LED Gruen

      digitalWrite(LedRot,   HIGH);                  // LED Rot ein
      digitalWrite(LedGruen, LOW);                   // LED Gruen aus

       if (digitalRead(Taster) == 0)                 // Taster Ein/Aus --> Groß/Klein
        {
          Grenzw_ACCEL_GradX = Altern_ACCEL_GradX;   // Alternative Empfindlichkeit
          Grenzw_ACCEL_GradY = Altern_ACCEL_GradY;
          Grenzw_GYRO_BesX   = Altern_GYRO_BesX;
          Grenzw_GYRO_BesY   = Altern_GYRO_BesY;
          Grenzw_GYRO_BesZ   = Altern_GYRO_BesZ;
        }
       else
        {
          Grenzw_ACCEL_GradX = Normal_ACCEL_GradX;   // Normale Empfindlichkeit
          Grenzw_ACCEL_GradY = Normal_ACCEL_GradY;
          Grenzw_GYRO_BesX   = Normal_GYRO_BesX;
          Grenzw_GYRO_BesY   = Normal_GYRO_BesY;
          Grenzw_GYRO_BesZ   = Normal_GYRO_BesZ;
        }

      Wire.begin();                                  // I2C aktivieren

      Wire.beginTransmission(MPU_addr);
      Wire.write(0x6B);                              // PWR_MGMT_1 Register
      Wire.write(0xff);                              // Reset MPU-6050
      Wire.endTransmission(true);
      delay(10);                                     // Wartezeit bis RESET ausgeführt
      Wire.beginTransmission(MPU_addr);
      Wire.write(0x6B);                              // PWR_MGMT_1 Register
      Wire.write(0x00);                              // MPU-6050 aktivieren
      Wire.endTransmission(true);
    }

//  ********************************************************************************************
```

Das Hauptprogramm

Im Hauptprogramm „void loop()“ werden Unterprogramme mit unterschiedlichen Aufgaben nacheinander aufgerufen. Dieser Ablauf wird ständig wiederholt.

```
//  ********************************************************************************

  void loop()
    {
      AuslesenMpu6050();                          // Accelerometer, Temperatur, Gyroscope
      FormatMpu6050();                            // Werte formatieren Accel, Temp, Gyro
      AbfrageTaster();                            // Tasterabfrage Alarmanlage Ein/Aus
      AlarmanlageAktivieren();                    // Alarmanlage verzoegert aktivieren
      Alarmauswertung();                          // Funktion Alarmauslösung
      Alarmausloesung();                          // Alarmauslösung Hupe ein
    }

//  ********************************************************************************
```

Die Funktion „void AuslesenMpu6050()“

Aus 14 Register des GY-521 werden die Werte Accelerometer, Temperatur und Gyroscope ausgelesen. Die Angaben der Zuweisung der Register sind dem Datenblatt der MPU-6050 entnommen. Im Datenblatt ist zu lesen, daß die ermittelten Werten die höchste Genauigkeit haben, wenn die Messeinrichtung horizontal ausgerichtet ist.

```
//  ********************************************************************************

void AuslesenMpu6050()                          // Accelerometer, Temperatur, Gyroscope
  {
    // Accelerometer, Temperatur und Gyroscope auslesen
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B);                           // Starten mit Register 0x3B (ACCEL_OutX_H)
    Wire.endTransmission(false);

    Wire.requestFrom(MPU_addr,14,true);         // Auslesen von 14 Register ab 0x3B
    ACCEL_OutX = Wire.read() <<8 | Wire.read(); // 0x3B (ACCEL_OutX_H) & 0x3C (ACCEL_OutX_L)
    ACCEL_OutY = Wire.read() <<8 | Wire.read(); // 0x3D (ACCEL_OutY_H) & 0x3E (ACCEL_OutY_L)
    ACCEL_OutZ = Wire.read() <<8 | Wire.read(); // 0x3F (ACCEL_OutZ_H) & 0x40 (ACCEL_OutZ_L)

    TEMP_Out   = Wire.read() <<8 | Wire.read(); // 0x41 (TEMP_Out_H)   & 0x42 (TEMP_Out_L)

    GYRO_OutX  = Wire.read() <<8 | Wire.read(); // 0x43 (GYRO_OutX_H)  & 0x44 (GYRO_OutX_L)
    GYRO_OutY  = Wire.read() <<8 | Wire.read(); // 0x45 (GYRO_OutY_H)  & 0x46 (GYRO_OutY_L)
    GYRO_OutZ  = Wire.read() <<8 | Wire.read(); // 0x47 (GYRO_OutZ_H)  & 0x48 (GYRO_OutZ_L)
  }

//  ********************************************************************************
```

Die Funktion „void FormatMpu6050()“

Die ausgelesenen Werte Accelerometer, Temperatur und Gyroscope werden hier formatiert. Die Faktoren für die Anpassung sind dem Datenblatt der MPU-6050 entnommen.

```
//  ********************************************************************************************

void FormatMpu6050()                              // Werte formatieren Accel, Temp, Gyro
  {
    // Werte Accelerometer (Beschleunigung) konvertieren zu 0 bis 2π RADIAN, Radian zu Degree
    // Funktionswert (arctan2) in einem Wertebereich von 360° (vier Quadranten) darstellen
    ACCEL_GradX = atan2(ACCEL_OutY, ACCEL_OutZ) * 180 / PI;
    ACCEL_GradY = atan2(ACCEL_OutX, ACCEL_OutZ) * 180 / PI;

    // Temperatur in °C umrechnen (Datenblatt)
    TEMP_Grad = TEMP_Out / 340.00 + 36.53;

    // Wert Gyroscope (Winkeldrehung) umrechnen
    GYRO_BesX = GYRO_OutX / 131.0 * 2;
    GYRO_BesY = GYRO_OutY / 131.0 * 2;
    GYRO_BesZ = GYRO_OutZ / 131.0 * 2;
  }

//  ********************************************************************************************
```

Die Funktion „void AbfrageTaster()“

Bei einem Signalwechsel (negative Flanke) am Reedkontakt wird die Alarmanlage aktiviert oder deaktiviert. Der aktuelle Zustand wird negiert.
Im weiteren werden hier die beiden Leuchtdioden für den Zustand „Alarmanlage Ein“ angesteuert und die Signaltöne für „Alarmanlage Ein“ und „Alarmanlage Aus“ angestoßen.

```
//  ********************************************************************************************

void AbfrageTaster()                              // Abfrage Taster Alarmanlage Ein/Aus
  {
    static byte TasterAlt;                        // Altwert Taster

    byte TasterNeu = digitalRead(Taster);         // Zustand Taster erfassen

     if  (TasterAlt == 1 && TasterNeu == 0 )      // Taster aktiviert (neg. Flanke)
      {
        Alarmanlage = !Alarmanlage;                           // Alarmanlage aus/ein
        digitalWrite(LedRot,   Alarmanlage == 0 ? HIGH:LOW);  // Aus/Ein, LED Rot   ein/aus
        digitalWrite(LedGruen, Alarmanlage == 1 ? HIGH:LOW);  // Ein/Aus, LED Gruen ein/aus

         if(Alarmanlage == 0)
          {
            AlarmanlageScharf = 0;                // Alarmanlage deaktivieren
            Alarm = 0;                            // Alarmauslösung abschalten
            TonAnlageAus();                       // Signalisierung Alarmanlage aus
          }
         else
          {
            StartZeitpunkt = millis();            // Zeitmessung Startzeit in ms
            TonAnlageEin();                       // Signalisierung Alarmanlage ein
          }

      }

    TasterAlt = TasterNeu;
  }

//  ********************************************************************************************
```

Unterprogramm Tonkennung „Alarmanlage Ein“

Ein dreimaliges Piepen signalisiert, Alarmanlage ist eingeschaltet.

```
//  ********************************************************************************************

void TonAnlageEin()                             // Ton Kennung Alarmanlage ein
  {
    for (byte i=0; i<3; i++)
      {
        tone(Summer, 2500, 250);
        delay(500);
      }
  }

//  ********************************************************************************************
```

Unterprogramm Tonkennung „Alarmanlage Aus“

Ein zweimaliges Piepen signalisiert, Alarmanlage ist ausgeschaltet.

```
//  ********************************************************************************************

void TonAnlageAus()                             // Ton Kennung Alarmanlage aus
  {
    for (byte i=0; i<2; i++)
      {
        tone(Summer, 1500, 50);
        delay(200);
      }
  }

//  ********************************************************************************************
```

Die Funktion „void AlarmanlageAktivieren()“

Die Alarmanlage wird nach den Einschalten („Alarmanlage Ein“) mit einer Zeitverzögerung aktiviert. Mit dem „Alarmanalge Scharf“ schalten wird der Signalton „Alarmanlage Scharf“ angestoßen.

```
//  ********************************************************************************************

void AlarmanlageAktivieren()                    // Alarmanlage verzoegert aktivieren
  {
     if (Alarmanlage == 1 && AlarmanlageScharf ==0 && millis()-StartZeitpunkt > 3000)
      {
        LageSichern();                          // aktuelle Werte Accel, Temp, Gyro sichern
        AlarmanlageScharf = 1;                  // Alarmanlage nach 3 Sekunden aktiviert
        TonAnlageScharf();                      // Signalisierung Alarmanlage scharf
      }
  }

//  ********************************************************************************************
```

Unterprogramm Tonkennung „Alarmanlage Scharf“

Ein einmaliges Piepen signalisiert, Alarmanlage ist scharf.

```
//  ********************************************************************************************

void TonAnlageScharf()                          // Ton Kennung Alarmanlage scharf
  {
    tone(Summer, 2100, 250);
  }

//  ********************************************************************************************
```

Unterprogramm „LageSichern()“

Kurz bevor die Alarmanlage in den Zustand „Alarmanlage Scharf“ übergeht, werden die aktuellen Werte Accelerometer, Temperatur und Gyroscope gesichert. Gleiches erfolgt auch nach der Beendigung einer Alarmauslösung.

```
//  ********************************************************************************************

void LageSichern()                              // aktuelle Werte Accel, Temp, Gyro sichern
  {
    Alt_ACCEL_GradX = ACCEL_GradX;              // Accelerometer X-Lage in Grad
    Alt_ACCEL_GradY = ACCEL_GradY;              // Accelerometer Y-Lage in Grad
    Alt_ACCEL_GradZ = ACCEL_GradZ;              // Accelerometer Z-Lage in Grad

    Alt_TEMP_Grad = TEMP_Grad;                  // Temperatur in °C

    Alt_GYRO_BesX = GYRO_BesX;                  // Gyroscope Beschleunigung X-Achse
    Alt_GYRO_BesY = GYRO_BesY;                  // Gyroscope Beschleunigung Y-Achse
    Alt_GYRO_BesZ = GYRO_BesZ;                  // Gyroscope Beschleunigung Z-Achse
  }

//  ********************************************************************************************
```

Die Funktion „void Alarmauswertung()“

Eine Winkeländerung oder Beschleunigung wird bei „Alarmanlage Scharf“ und nicht anstehender „Alarmauslösung“ überwacht und führt bei Grenzwertverletzung zur Alarmauslösung. Die Abläufe bei einer Alarmauslösung sind in der Funktion „void Alarmauslösung()“ hinterlegt.

```
//  ********************************************************************************************

void Alarmauswertung()                          // Funktion Alarmauslösung
  {
     if (AlarmanlageScharf == 1 && Alarm != 1)    // Alarmanlage ist scharf, kein Alarm
      {
        // Alarmauslösung bei Winkeländerung X oder Y um x-Grad oder Beschleunigung
         if (abs(Alt_ACCEL_GradX - ACCEL_GradX) > Grenzw_ACCEL_GradX  ||
             abs(Alt_ACCEL_GradY - ACCEL_GradY) > Grenzw_ACCEL_GradY  ||
             abs(Alt_GYRO_BesX - GYRO_BesX)     > Grenzw_GYRO_BesX    ||
             abs(Alt_GYRO_BesY - GYRO_BesY)     > Grenzw_GYRO_BesY    ||
             abs(Alt_GYRO_BesZ - GYRO_BesZ)     > Grenzw_GYRO_BesZ)
          {
            StartZeitpunkt = millis();          // Zeitmessung Startzeit in ms
            Alarm = 1;                          // Alarmauslösung Hupe ein
          }
      }
  }

//  ********************************************************************************************
```

Die Funktion „void Alarmauslösung“

Bei einer Alarmauslösung werden die rote Leuchtdiode, die Ansteuerung für das Hupenmodul und der Buzzer (Software Intervall) aktiviert. Nach Ablauf der Alarmzeit werden diese deaktiviert.

```
//  ********************************************************************************************

void Alarmausloesung()                          // Alarmauslösung Hupe ein
  {
     if (Alarm == 1)
      {
        tone(Summer, 2000, 150);
        delay(250);

         if (millis()-StartZeitpunkt > 30000)
          {
            Alarm = 0;                          // Alarmauslösung Zeit abgelaufen
            LageSichern();                      // aktuelle Werte Accel, Temp, Gyro sichern
          }

        digitalWrite(LedRot,   Alarm == 1 ? HIGH:LOW);  // Ein/Aus, LED Rot   ein/aus
        digitalWrite(Hupe,     Alarm == 1 ? HIGH:LOW);  // Ein/Aus, Intervall Piezo  ein/aus
      }
  }

//  ********************************************************************************************
```

Schlußbemerkung

Diese kleine Alarmanlage sehe ich als ein mögliches Beispiel für den Umgang mit Hard- und Software zu praktizieren. Dabei kann dann ein sinnvolles Gerät nach eigenen Vorstellungen kreiert werden. Da jeder eigene Vorstellungen hat, kann er diese in dieses Beispiel einfließen lassen oder ein anderes Programm mit völlig anderen Funktionen schreiben.