Documentation du Système d'avertissement d'altitude basse (GPWS)

Louis Hermier

23/09/2020

Contenu

Introduction et motivation

Le but est d'avoir un circuit basé sur une carte Arduino qui permet d'indiquer rapidement de manière très simple au pilote ou à des utilisateurs extérieurs un ordre de grandeur de l'altitude du drone sans avoir à détourner les yeux de ce dernier. Le circuit comporte trois LEDs: Une led qui indique le fonctionnement du circuit[1], une LED allumée lorsque le drone est à une altitude considérée comme nécessitant une attention élevé de la part du pilote[2], et une LED allumée lorsque le drone est considéré comme étant à une altitude dangereusement proche du sol[3]. L'altitude du drone est obtenue par (Attente du capteur). Le choix de la carte Arduino permet d'avoir un circuit qui est déjà en fonctionnement et initialisé quelques secondes au maximum après avoir mis le drone sous tension, contrairement à par exemple un SBC[4] comme le Raspberry Pi, qui propose plus de puissance de calcul (dont le circuit n'a pas besoin), aux prix d'une consommation électrique plus élevée et d'un temps de démarrage (OS complet) plus conséquent.

Le code source complet peut être trouvé sur le **dépot GitHub**[5] du projet dans le dossier `GPWS/`.

---

[1] `STATUS`
[2] `WARNING`
[3] `DANGER`
[4] Single Board Computer - litérallement "Ordinateur sur une seule planche"
[5] **https://github.com/LouisH-760/Drone**

Utilisation des pins

Entrées

(Attente du Capteur)

Sorties

`STATUS`

Le pin `STATUS` est placé par défaut sur le **pin numérique 4**. Il correspond à la LED verte, qui est allumée en permanence[6] pour montrer que le circuit est alimenté.

`WARNING`

Le pin `WARNING` est placé par défaut sur le **pin numérique 2**. Il correspond à la LED jaune, qui s'allume lorsque le drone est entre deux altitudes définies, considérées comme zone d'"avertissement" dans laquelle la concentration du pilote doit être renforcée.

`DANGER`

Le pin `DANGER` est placé par défaut sur le **pin numérique 3**. Il correspond à la LED rouge, qui s'allume lorsque le drone est entre deux altitudes définies (avec une généralement le sol), dans une zone considérée comme une zone de danger.

Résumé

Remarque: seuls les pins utilisés sont présents.

Numéro	Type	Nom
2	sortie	`WARNING`
3	sortie	`DANGER`
4	sortie	`STATUS`

[6] Sauf en cas de malfonction

Constantes

Constantes de configuration des pins

```
#define WARNING 2
#define DANGER 3
#define STATUS 4
```

Les trois constantes:

1. `WARNING`
2. `DANGER`
3. `STATUS`

Servent à indiquer quels pins correspondent aux fonctions décrites précédemment.

Constantes de configuration des délais

```
#define TEST_DELAY 100
#define MALFUNC_DELAY 500
```

Ces constantes servent à configurer les différents délais utilisés dans le code pour permettre que les différent états des LEDs soient observables par l'utilisateur.

- `TEST_DELAY` correspond au délai utilisé dans la fonction de test pour les LEDs ou autres sorties digitales. La valeur est un entier, correspondant à un temps en millisecondes. La valeur par défaut est de 100 millisecondes. Cela correspond à la durée d'allumage de la LED testée et au temps de pause entre chaque test.
- `MALFUNC_DELAY` correspond au délai Allumé/Éteint des LEDs spécifiées lors de l'appel à la fonction de malfonction. La valeur est un entier, correspondant à un temps en millisecondes. La valeur par défaut est de 500 millisecondes. Cela correspond à la durée d'allumage des LEDs données et au temps de pause entre chaque allumage.

Constantes de configuration des altitudes

```
#define GROUND 0
#define MAX_DANGER 20
#define MAX_WARNING 50
#define MAX_START_HEIGHT 10
#define MIN_REAL_HEIGHT 0
```

Ces constantes servent à définir les paliers pour les zones "Danger" et "Warning" en fonction des valeurs renvoyées par le capteur.

- `GROUND` correspond à une valeur que l'on va considérer comme le sol, et sert de limite minimum à la zone "Danger".
- `MAX_DANGER` correspond à la limite supérieure de la zone "Danger" (et donc à la limite inférieure de la zone "Warning").
- `MAX_WARNING` correspond à la limite supérieure de la zone "Warning".
- `MAX_START_HEIGHT` correspond à la valeur d'altitude au démarrage au dessus de laquelle le programme va considérer que le capteur est défectueux et passer en mode "malfonction".
- `MIN_REAL_HEIGHT` correspond à une valeur donnée par le capteur en dessous de laquelle le programme va considérer que le capteur est défectueux et passer en mode "malfonction".

Variables Globales

```
int height;
```

La seule variable présente est un entier, servant à stocker l'altitude mesurée par le capteur. `height` est une variable globale, car elle est utilisée dans la fonction `void loop()`, et devoir la recréer à chaque fois serait trop coûteux. Cette variable n'est pas évitable avec le code existant.

Fonctions

`void test(int count, ...)`

```
void test(int count, ...) {
  va_list args;
  int current;
  va_start(args, count);
  for(int i = 0; i < count; i++) {
    current = va_arg(args, int);
    digitalWrite(current, HIGH);
    delay(TEST_DELAY);
    digitalWrite(current, LOW);
    delay(TEST_DELAY);
  }
  va_end(args);
}
```

Cette fonction sert à tester des sorties (Prévue pour des LEDs à la base). Elle prend un nombre variable d'arguments de type entier (`int`). Ces arguments se séparent en deux parties:

1. Une partie fixe: un argument de type entier, qui donne le nombre d'arguments passés dans la partie variable.
2. Une partie variable: un nombre variable d'arguments de type entier, qui indique les numéros de pins à tester.

Exemple d'utilisation:

```
test(3, WARNING, DANGER, STATUS);
```

Fonctionnement:

- La fonction parcourt la liste d'arguments passés du début à la fin
- Pour chaque pin, on a:
  - Le pin est passé à `HIGH` ("Allumé") à l'aide de `digitalWrite()`
  - on attend le délai spécifié dans la constante `MALFUNC_DELAY`
  - Le pin est passé à `LOW` ("Éteint") à l'aide de `digitalWrite()`
  - on attend le délai spécifié dans la constante `MALFUNC_DELAY`
- la liste d'arguments variables est libérée pour économiser la mémoire

`void malfunction(int beginPin, int endPin)`

```
void malfunction(int beginPin, int endPin) {
  while(true) {
      for(int i = beginPin; i <= endPin; i++) {
          digitalWrite(i, HIGH);
      }
      delay(MALFUNC_DELAY);
      for(int i = beginPin; i <= endPin; i++) {
        digitalWrite(i, LOW);
      }
      delay(MALFUNC_DELAY);
  }
}
```

La fonction sert à notifier l'utilisateur d'une malfonction possible du capteur. Une fois lancée, elle bloque le reste du programme et se contente de faire "clignoter" les LEDs de l'intervalle spécifié. Attention, tous les pins entre `beginPin` et `endPin` doivent être des **sorties**, idéalement attachés à des LEDs!

La fonction a deux arguments:

1. `beginPin` (`int`) qui précise le pin (inclus) le plus petit attaché à une LED
2. `endPin` (`int`) qui précise le pin (inclus) le plus grand attaché à une LED

Exemple d'utilisation:

```
if(height < MIN_REAL_HEIGHT) {
  malfunction(WARNING, STATUS);
}
```

Fonctionnement:

- le code est contenu dans une boucle infinie[7] pour bloquer complètement l'exécution: seul un appui sur le bouton `reset` ou un retrait de l'alimentation permet d'en sortir. Cela permet de signifier la malfonction sans ambiguïté à l'utilisateur et nécessite une action de celui-ci pour retourner à un mode de fonctionnement normal.
  - à l'aide d'une boucle `for`, tous les pins sont mis à `HIGH`
  - on attend le délai spécifié par `MALFUNC_DELAY`
  - à l'aide d'une boucle `for`, tous les pins sont mis à `LOW`
  - on attend le délai spécifié par `MALFUNC_DELAY`
  - etc…

`void updateHeight(int *h)`

```
void updateHeight(int *h) {
  // à compléter
}
```

La fonction sert à mettre à jour le contenu de la variable spécifiée à l'aide du capteur à Ultrasons. La fonction prend un seul argument:

1. `int *h` (pointeur vers `int`): pointeur vers la variable à mettre à jour, de type entier.

Exemple d'utilisation:

```
updateHeight(&height);
```

Fonctionnement:

(section vide)

`void setup()`

```
void setup() {
  pinMode(WARNING, OUTPUT);
  pinMode(DANGER, OUTPUT);
  pinMode(STATUS, OUTPUT);
  test(3, WARNING, DANGER, STATUS);

  updateHeight(&height);
  if(height > MAX_START_HEIGHT || height < MIN_REAL_HEIGHT) {
    malfunction(WARNING, STATUS);
  }
  digitalWrite(STATUS, HIGH);
}
```

Cette fonction est la fonction d'initialisation commune à tous les programmes Arduino. Les modes pour les pins y sont généralement définis. Cette fonction ne prend pas d'arguments. Dans notre programme, des tests sont aussi fait pour assurer un bon fonctionnement du système et la LED `STATUS` est allumée pour montrer le bon fonctionnement du système.

`void loop()`

```
void loop() {
  updateHeight(&height);
```

[7] ici, un simple `while(true) { ... }`

```
  if(height < MIN_REAL_HEIGHT) {
    malfunction(WARNING, STATUS);
  }
  if(height <= MAX_DANGER) {
    digitalWrite(WARNING, LOW);
    digitalWrite(DANGER, HIGH);
  } else if(height > MAX_DANGER && height <= MAX_WARNING) {
    digitalWrite(DANGER, LOW);
    digitalWrite(WARNING, HIGH);
  } else {
    digitalWrite(WARNING, LOW);
    digitalWrite(DANGER, LOW);
  }
}
```

Cette fonction est la fonction qui fait office de boucle principale dans tous les programmes Arduino. Ici, son fonctionnement est assez simple, et comporte deux étapes:

1. Acquisition de données
   - Cela est fait à l'aide de `updateHeight()`.
2. Traitement des données
   - D'abord, la validité du résultat est testé. S'il ne correspond pas à un résultat normal, la fonction `malfunction()` est appelée.
   - Puis, les données sont utilisées pour allumer (ou non) la LED attendue en fonction de l'altitude, à l'aide de `digitalWrite`.