Drone Simulation Design Manual

Author: Jack Jiang (z5129432)

Date: 03/10/2017

This design manual describes how the emulating system is designed, includes:

1. Introduction of code
2. Module description
3. Flowchart diagram
4. Data structures of mountain map
5. Algorithms of search and return

1. Introduction:

Program entry and subroutine:

➔ The code will begin from ***RESET***, do some initialisation jobs and then come to ***MAIN.***

➔ External Interrupt 1 is connected to the left bottom, used to start the mission after a valid input is got. The related subroutine is ***LEFT_INT***.

➔ External Interrupt 0 is connected to the left bottom, used to start the mission while searching. The related subroutine is ***RIGHT_INT***.

Modules ***MAIN, LEFT_INT*** and ***RIGHT_INT*** will be described in ***chapter 2: Module description***

The sequence of each module will be described in ***chapter 3: Flowchart diagram***

Global variables:

➔ Register ACCIDENT_X: store accident point x from user input

➔ Register ACCIDENT_Y: store accident point y from user input

➔ Register GLOBAL1: keep track of current point x of drone

➔ Register GLOBAL2: keep track of current point y of drone

Included inc file:

The map of the mountain is stored in program memory. A separate *.inc file is included to store the data by using .INCLUDE directive.

Details of data structures of this *.inc file will be described in ***chapter 4: Mountain map data structure***.

Drone configuration:

```
;============ configuration =============
        .EQU     DRONE_SPEED =  2000
        .EQU     DRONE_HEIGHT = 9
;=====================================
```

The searching speed of the drone can be easily configured by changing the value of ***DRONE_SPEED***, which refer to the time spent on flying 1 meter. The unit of time is 0.1 millisecond.

For example:

if we set ***DRONE_SPEED*** = 2000, the drone will spend 0.2 seconds to fly 1 meter.

In the worst case, the longest search path is 64*64 = 4096 meter. Therefore, the longest search time is:

$$DRONE_{¿}SPEED/10000*4096\,seconds$$

Which is 819 second in this case.

The height above the ground when flying can be easily configured by changing the value of ***DRONE_HEIGHT.***

Search path and the calculation of flying height will be described ***in chapter 5: Algorithms of search and return.***

2. Module description:

MAIN:

Get the accident point through the keypad and check the validity of it. After # is pushed, enable INT1(left button), and standby.

LEFT_INT:

This module can be divided into three part:

DRONE_TAKE_OFF:

After the left button is pushed, start to take-off, in which drone height will gradually increase to ***Flying height***.

DRONE_SEARCH:

Search the accident point and display the current location to LCD. When finding the target, the drone will go into inspecting mode.

INT2(right button) will be enabled before searching and disable after searching, which means we can only abort the mission by push right button while searching.

DRONE_INSPECT:

When finding the target, the drone will halt for a while and then goes into DRONE_RETURN.

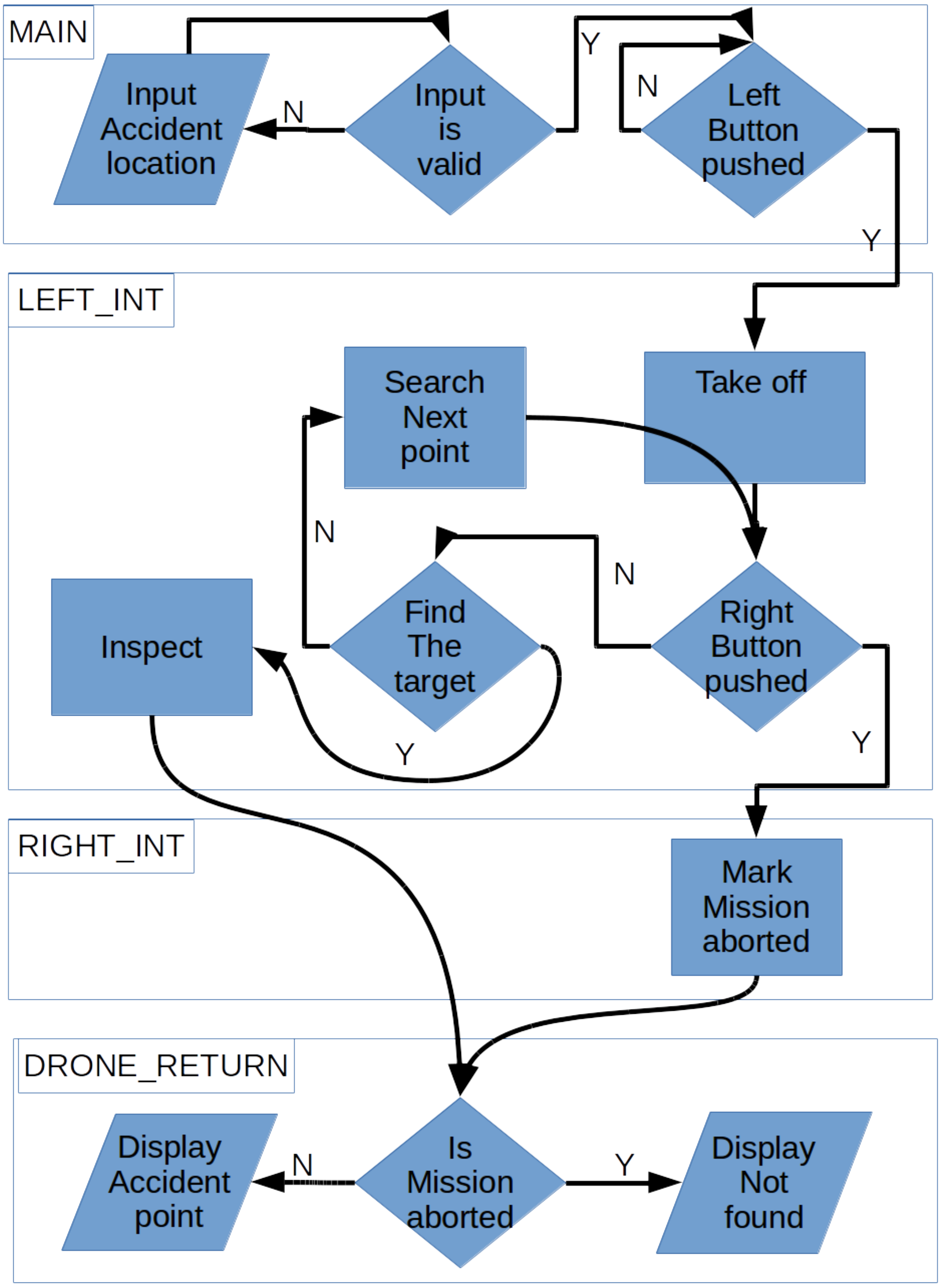
RIGHT_INT:

This module can only be run when the right button is pushed during the search. It only gives a mark that the mission was aborted and the target hasn't been found. After that, the drone will go into DRONE_RETURN.

DRONE_RETURN:

The drone will return following the return path and then land on the ground. After that, accident point will be shown if the target has been found. If the mission is aborted, it will display "not found" instead.

3. Flowchart diagram:



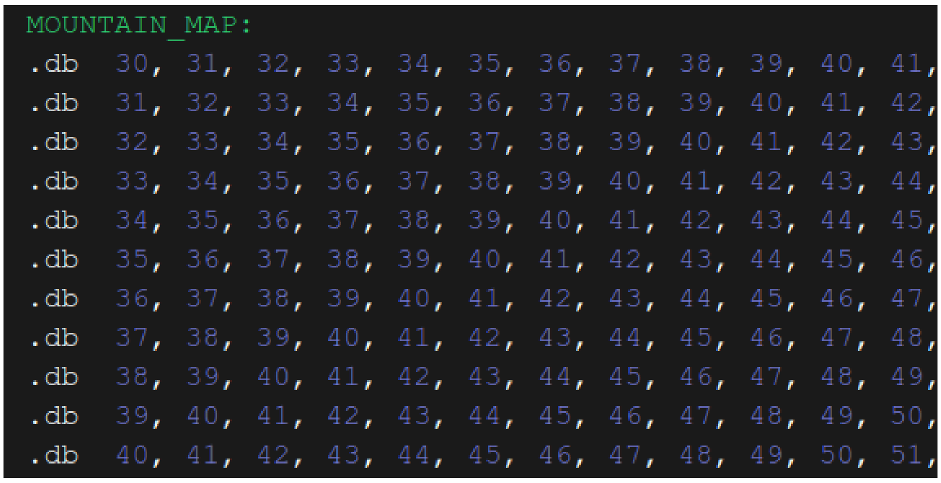
4. Mountain map data structures:

The (x, y, z) of every point in the mountain can be derived from the *.inc file.

The (x, y) of each point is determined by the sequence of those data.

(0, 0)	(0, 1)	(0, 2)	…	(0, 61)	(0, 62)	(0, 63)
(1, 0)	(1, 1)	(1, 2)	…	(1, 61)	(1, 62)	(1, 63)
(2, 0)	(2, 1)	(2, 2)	…	(2, 61)	(2, 62)	(2, 63)
…	…	…	…	…	…	…
(61, 0)	(61, 1)	(61, 2)	…	(61, 61)	(61, 62)	(61, 63)
(62, 0)	(62, 1)	(62, 2)	…	(62, 61)	(62, 62)	(62, 63)
(63, 0)	(63, 1)	(63, 2)	…	(63, 61)	(63, 62)	(63, 63)

The z (height) of each point is stored in the *.inc file, using 1 byte for each point, starting from a label called MOUNTAIN_MAP.

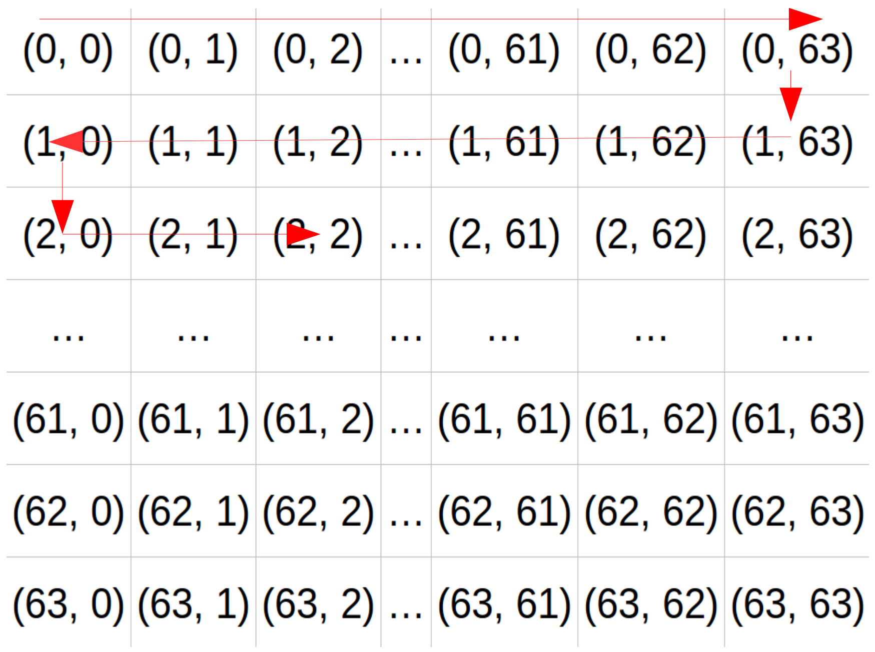


```
MOUNTAIN_MAP:
.db  30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
.db  31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
.db  32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
.db  33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
.db  34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
.db  35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
.db  36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
.db  37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
.db  38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
.db  39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
.db  40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
```

5. Algorithms of search and return:

Search path:

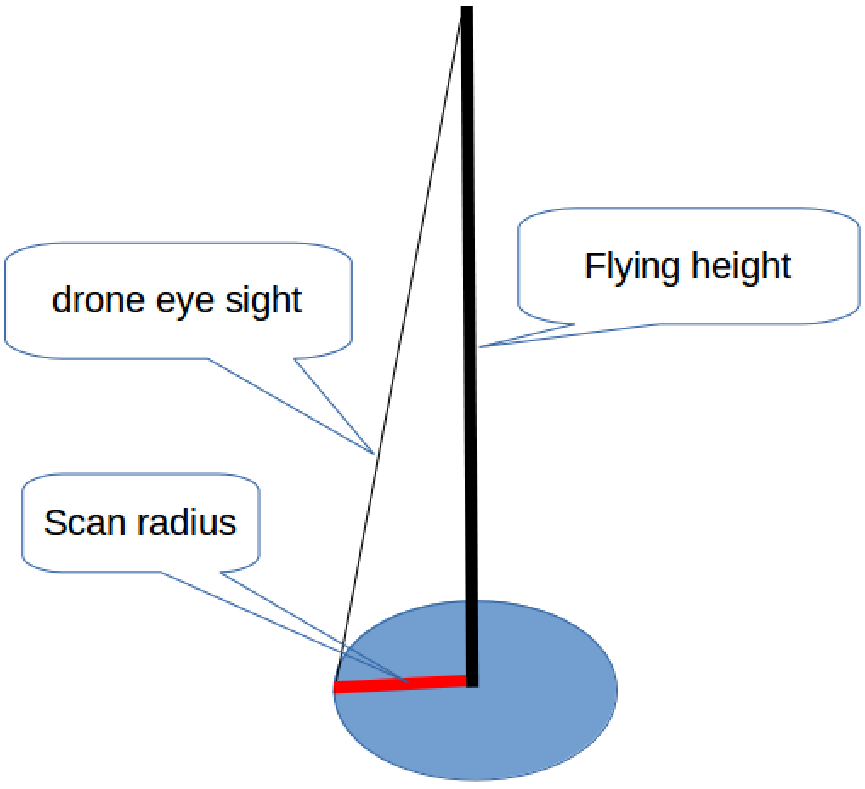
The drone will fly as the following path.



(0, 0)	(0, 1)	(0, 2)	…	(0, 61)	(0, 62)	(0, 63)
(1, 0)	(1, 1)	(1, 2)	…	(1, 61)	(1, 62)	(1, 63)
(2, 0)	(2, 1)	(2, 2)	…	(2, 61)	(2, 62)	(2, 63)
…	…	…	…	…	…	…
(61, 0)	(61, 1)	(61, 2)	…	(61, 61)	(61, 62)	(61, 63)
(62, 0)	(62, 1)	(62, 2)	…	(62, 61)	(62, 62)	(62, 63)
(63, 0)	(63, 1)	(63, 2)	…	(63, 61)	(63, 62)	(63, 63)

It can be known that the distance between two paths is 1 meter. In order to cover all the point on the surface of the mountain, the ***Scan radius*** of drone must be larger than 0.5 meters.

Scan radius:



When flying in the sky, the area which the drone can see is a circle, which radius is defined as ***Scan radius***.

From the above diagram, we know that:

$$Flying\,Height^2 + Scan\,radius^2 = Drone\,eyes\,sight^2$$

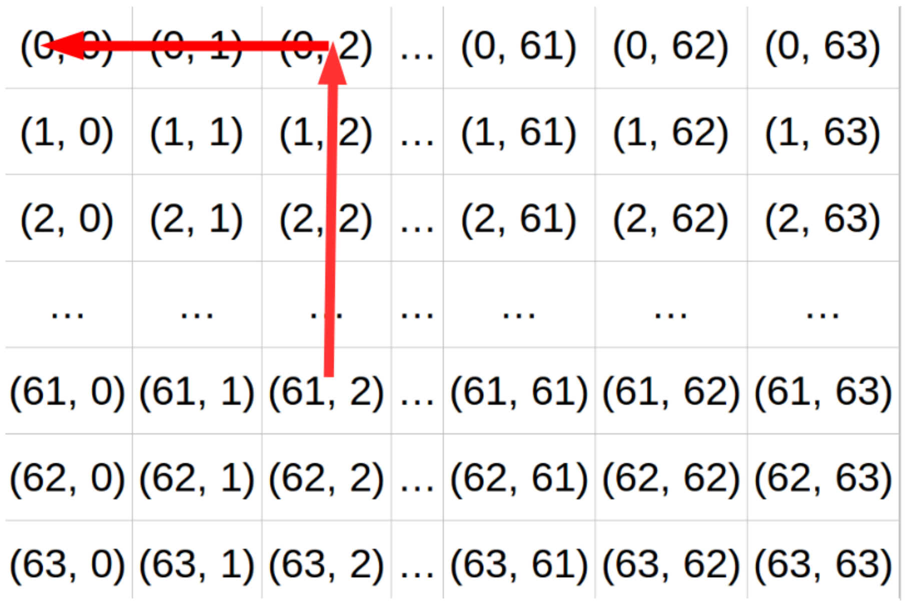
Assume that:

- ***Drone eyes sight*** = 10m
- ***Flying height*** = 9m

Therefore, the ***Scan radius*** is much more than 0.5 meter.

Return path:

When drone find the target or the mission is aborted, the drone will return as the following path:



(0, 0)	(0, 1)	(0, 2)	…	(0, 61)	(0, 62)	(0, 63)
(1, 0)	(1, 1)	(1, 2)	…	(1, 61)	(1, 62)	(1, 63)
(2, 0)	(2, 1)	(2, 2)	…	(2, 61)	(2, 62)	(2, 63)
…	…	…	…	…	…	…
(61, 0)	(61, 1)	(61, 2)	…	(61, 61)	(61, 62)	(61, 63)
(62, 0)	(62, 1)	(62, 2)	…	(62, 61)	(62, 62)	(62, 63)
(63, 0)	(63, 1)	(63, 2)	…	(63, 61)	(63, 62)	(63, 63)