square_demo_code

```
//this code will make the SPHERES satellite navigate in a square.

//Declare any variables shared between functions here
float corners[4][3];
int currentCorner;

void init(){
    //This function is called once when your code is first loaded.
    //IMPORTANT: make sure to set any variables that need an initial value.
    //Do not assume variables will be set to 0 automatically!
    float corner0[3] = { 0.6,  0.6, -0.5};
    float corner1[3] = { 0.6, -0.6, -0.5};
    float corner2[3] = {-0.6, -0.6, -0.5};
    float corner3[3] = {-0.6,  0.6, -0.5};
    memcpy(corners[0], corner0, 3*sizeof(float));
    memcpy(corners[1], corner1, 3*sizeof(float));
    memcpy(corners[2], corner2, 3*sizeof(float));
    memcpy(corners[3], corner3, 3*sizeof(float));
    currentCorner = 0;
}   //--- fine di init

void loop(){
    //This function is called once per second.  Use it to control the satellite.

    if (currentCorner == 0) {
        if (isCloseTo(corners[0])) {
            currentCorner = 1;
        } else {
            api.setPositionTarget(corners[0]);
        }
    }
    if (currentCorner == 1) {
        if (isCloseTo(corners[1])) {
            currentCorner = 2;
        } else {
            api.setPositionTarget(corners[1]);
        }
    }
    if (currentCorner == 2) {
        if (isCloseTo(corners[2])) {
            currentCorner = 3;
        } else {
            api.setPositionTarget(corners[2]);
        }
    }
```

```
if (currentCorner == 3) {
        if (isCloseTo(corners[3])) {
            currentCorner = 0;
        } else {
            api.setPositionTarget(corners[3]);
        }
    }

}    //---  fine di loop

bool isCloseTo (float *target) {
    ZRState myState;
          api.getMyZRState(myState);

    float distanceSquared = (target[0] - myState[0]) * (target[0] - myState[0])
                          + (target[1] - myState[1]) * (target[1] - myState[1])
                          + (target[2] - myState[2]) * (target[2] - myState[2]);
    return distanceSquared < 0.01;
}
```

---

init

Definiti i 4 punti `corner0[3]` ... `corner3[3]` (array monodimensionali di 3 elementi di tipo float) da raggiungere sul piano z= -0.5, viene inizializzata la matrice 4*3 `corners[4][3]` (array bidimensionale che rappresenta 4 punti di 3 coordinate) mediante la funzione `memcpy` .

http://www.cplusplus.com/reference/cstring/memcpy/

`currentCorner` viene inizializzato a zero, in modo da dirigersi in *loop* inizialmente verso `corners[0]` .

loop

Ogni secondo la funzione fa dirigere lo SPHERES verso il `corners[i]` opportuno (**i** `= currentCorner`), ma se lo SPHERES raggiunge il punto indicato da `currentCorner`, la variabile `currentCorner` viene incrementata (oppure viene posta a zero al raggiungimento del 4^ punto, `corner[3]`).

Lo SPHERES raggiunge il punto indicato alla distanza ritenuta opportuna per questo programma (1 dm) se il quadrato della distanza dal punto diventa minore di `0.01`, in tal caso la funzione booleana **isCloseTo** restituisce il valore `true` (l'istruzione `return distanceSquared < 0.01` restituisce `true` se viene soddisfatta la condizione `distanceSquared < 0.01` , `false` altrimenti).

isCloseTo

Determina la posizione dello SPHERES impostando l'array `myState`, calcola in `distanceSquared` il quadrato della distanza dal punto `target[]` (parametro passato dalla funzione chiamante, quindi assume di volta in volta le coordinate di `corner[0]`...`corner[3]`, restituisce un valore boolenano.