

See references for
internal information

```
1 // Use DBML to define your database structure
2 // Docs: https://dbml.dbdiagram.io/docs
3
4
5
6 Table users {
7   id integer [primary key]
8   username varchar
9   email varchar
10  password varchar
11  num_drones int [note: 'Amount of drones']
12  role int [note: 'Privileges']
13  session_at timespamp [note: 'Last logged time']
14  created_at timestamp [note: 'Account creation time']
15 }
16
17 Table roles {
18   id int [primary key]
19   description varchar
20   name varchar
21   level int [note: 'Privilegy level (Low 0 - 9 High)']
22 }
23
24 Table drones {
25   dic varchar [primary key]
26   auth_code varchar
27   metadata json
28   owner array
29   status boolean
30   flight blob
31   registered_at timestamp [note: 'First registration date']
32   imported_at timestamp [note: 'First time imported to platform']
33 }
34
35 Ref: drones.owner > users.id
36 Ref: users.role > roles.id
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
```

drone-front.db

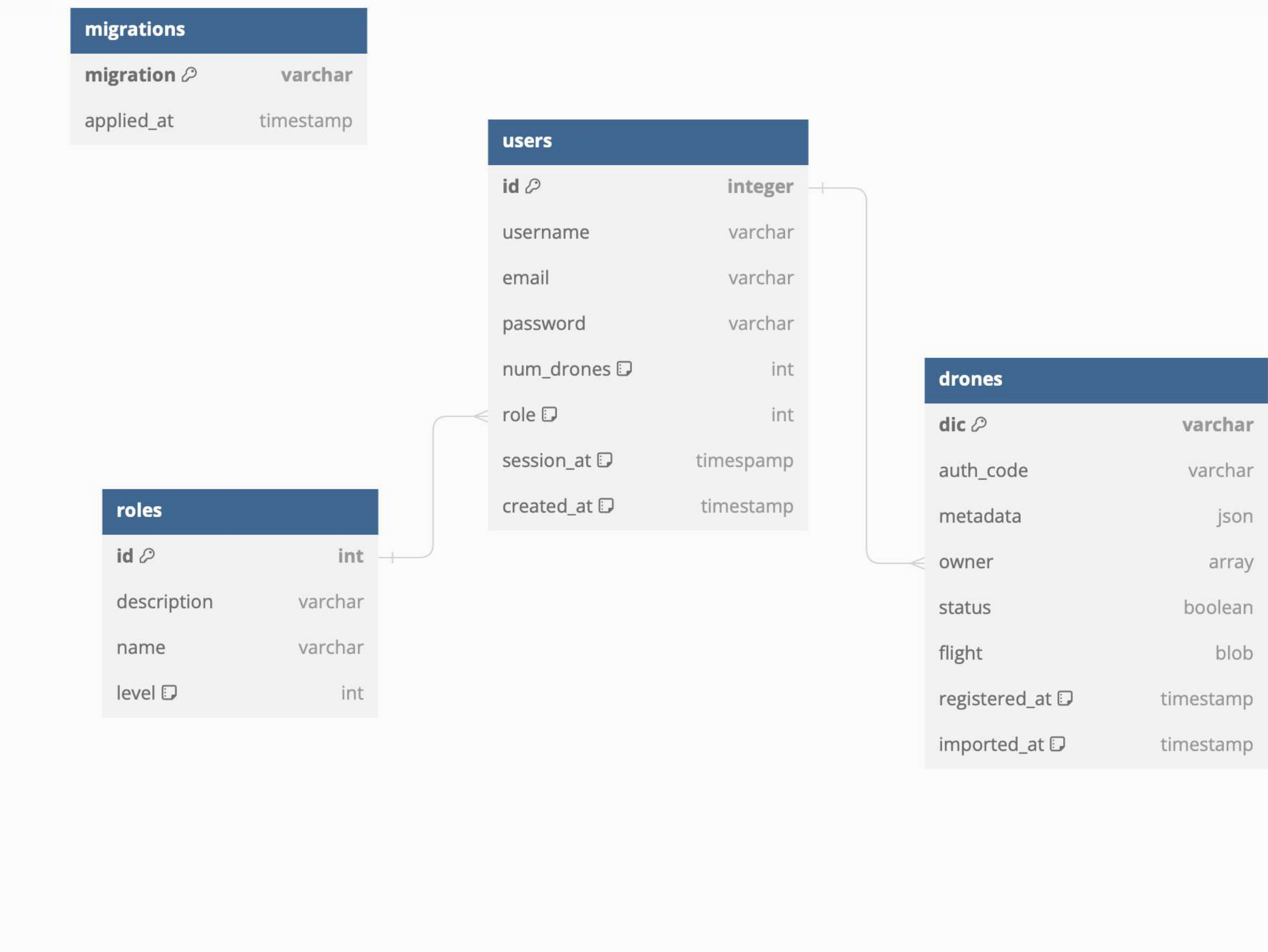


migrations	
migration	varchar
applied_at	timestamp

users	
id	integer
username	varchar
email	varchar
password	varchar
num_drones	int
role	int
session_at	timespamp
created_at	timestamp

drones	
dic	varchar
auth_code	varchar
metadata	json
owner	array
status	boolean
flight	blob
registered_at	timestamp
imported_at	timestamp

roles	
id	int
description	varchar
name	varchar
level	int

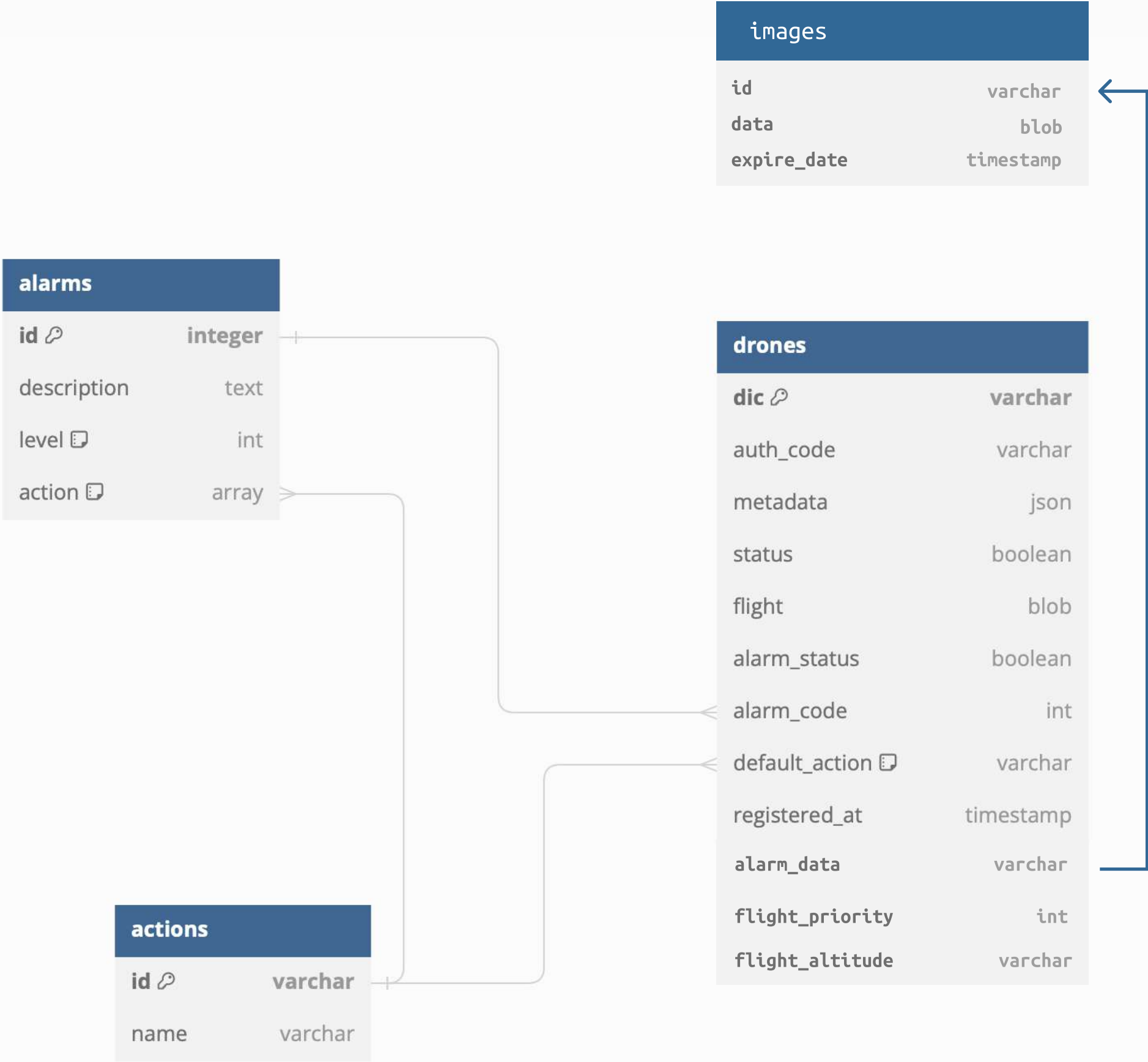


```
1 // Use DBML to define your database structure
2 // Docs: https://dbml.dbdiagram.io/docs
3
4 Table migrations {
5   migration varchar [primary key]
6   applied_at timestamp
7 }
8
9 Table alarms {
10  id integer [primary key]
11  description text
12  level int [note: 'From 0 to 9 (Low to High)']
13  action array [note: 'Array of actions to follow']
14 }
15
16 Table actions {
17  id varchar [primary key]
18  name varchar
19 }
20
21 Table drones {
22  dic varchar [primary key]
23  auth_code varchar
24  metadata json
25  status boolean
26  flight blob
27  alarm_status boolean
28  alarm_code int
29  default_action varchar [note: 'Default action set by drone in case critic']
30  registered_at timestamp
31 }
32
33 Ref: drones.alarm_code > alarms.id
34 Ref: alarms.action > actions.id
35 Ref: drones.default_action > actions.id
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
```

ai_entities	
id	varchar
server_ip	varchar
server_port	int
capacity	int
max_capacity	int

migrations	
migration	varchar
applied_at	timestamp

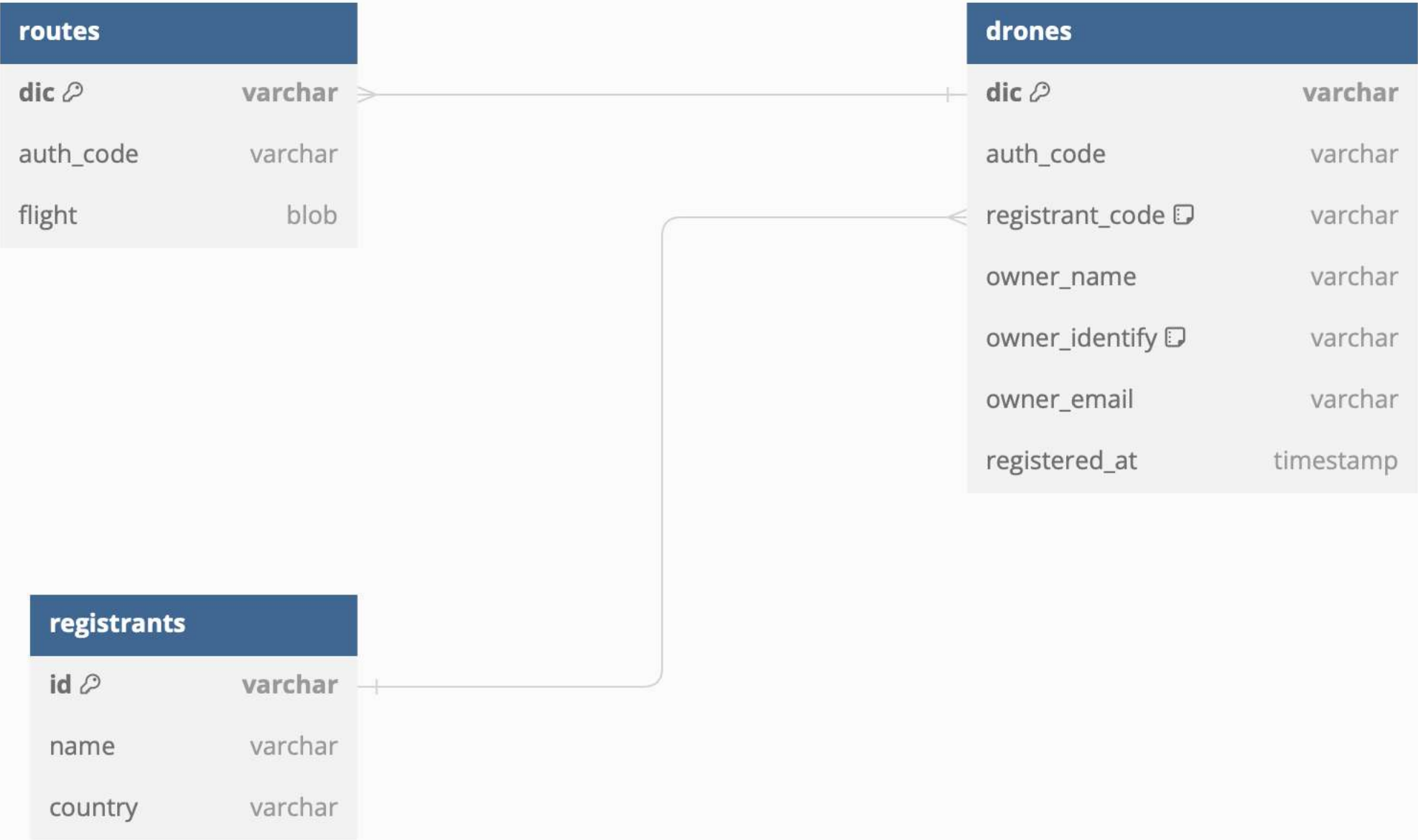
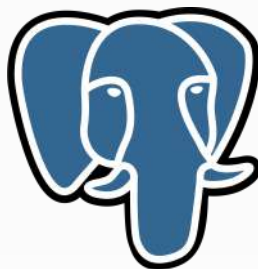
drone-module.mongo




```
1 // Use DBML to define your database structure
2 // Docs: https://dbml.dbdiagram.io/docs
3
4 Table routes {
5   dic varchar [primary key]
6   auth_code varchar
7   flight blob
8 }
9
10 Table registrants {
11   id varchar [primary key]
12   name varchar
13   country varchar
14   // More data, but not going to use in v1
15 }
16
17 Table drones {
18   dic varchar [primary key]
19   auth_code varchar
20   registrant_code varchar [note: 'Any registrand platform code']
21   owner_name varchar
22   owner_identify varchar [note: 'Any identifiable document']
23   owner_email varchar
24   registered_at timestamp
25 }
```

```
27 Ref: drones.registrant_code > registrants.id
28 Ref: routes.dic > drones.dic
```

core-system.db



1.

Funcionalidades del frontd

- Listar drones / minimal info de cada drone
- Mapa de todo los drones
- Crear rutas y ver rutas del drone
- Ver las alarmas, con imagen evidencia

Extra Información para todos

Las rutas creadas por el cliente son conjunto de coordenadas en formato **Grado decimal**.

Por ejemplo: [(1,2), (2,3), (3,4)]

Entre coordenada a coordenada no puede existir un gap mayor que 5m. Es decir 1/4 segundos en Europa

API Proporcionadas por el server

Toda peticion deberá llevar su auth_code en header **Authorization**

<dic> : Matricula del drone. Para obtener varias, separar por ;
Ejemplo: ESP12345-001-0001;ESP12340-002-0001

Y auth-code separadas por ; por el mismo orden de <dic>

Los datos en JSON se devolverá con key de la matricula del drone

GET /metadata/[all, battery, gps, smoke, altitude, orientation]/<dic>

GET /status/<dic>

GET /route/<dic>

POST /route/<dic> -----> Body: Conjunto de coordenadas

GET /alarm/<dic>

GET /alarm/image/<id>

```
{
  status: "error",
  message: "401 Unauthorized",
  path: "/metadata"
  errcode: 401
}

{
  status: "ok",
  path: "/status/ESP00001-001-9999;ESP00001-002-9999"
  data: {
    "ESP00001-001-9999": {
      status: 0
    },
    "ESP00001-002-9999": {
      status: 1
    }
  }
}

{
  status: "ok",
  path: "/alarm/ESP00001-001-9999"
  data: {
    "ESP00001-001-9999": {
      status: 1,
      image: {
        path: "/img/tmpESP00001-001-9999.png"
      }
    }
  }
}
```

Interacción con DDBB

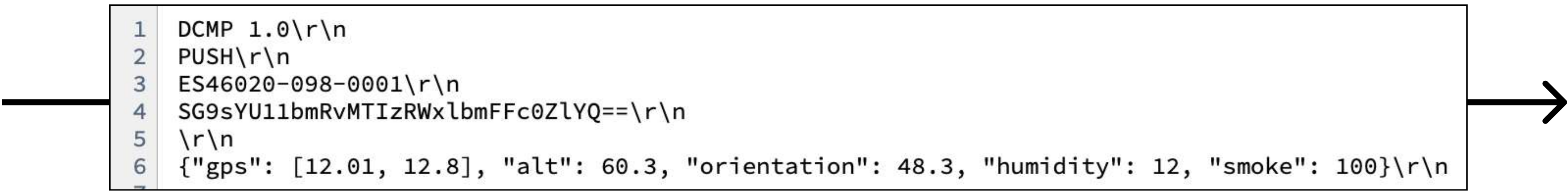
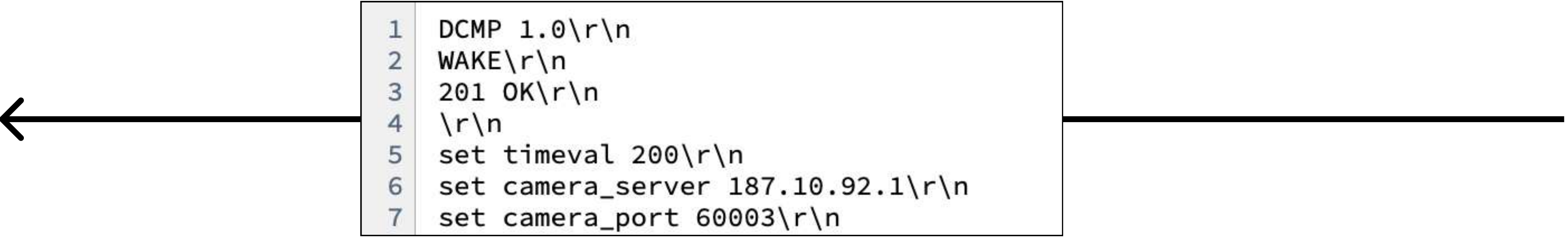
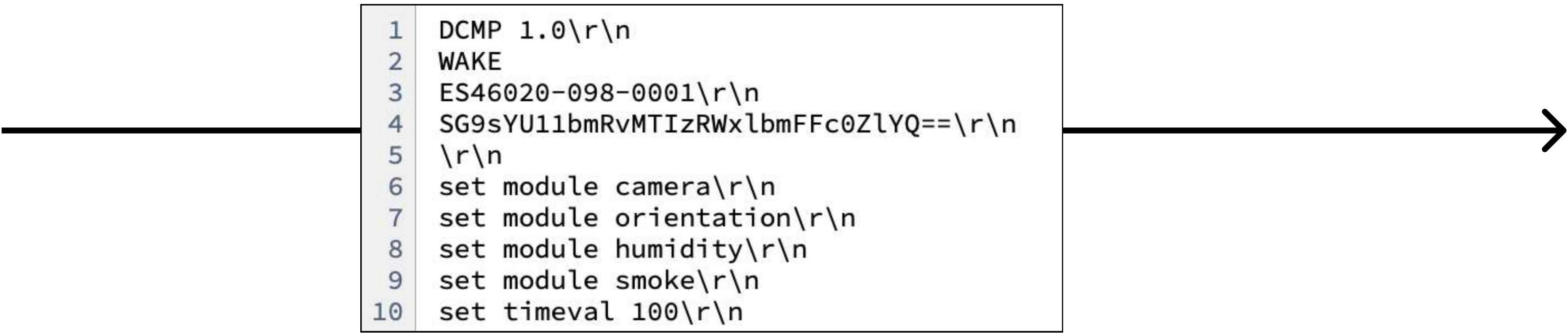
- Guardar Coordenadas
- Recoger metadatos
- Recoger status
- Recoger alarma y crear archivo con evidencia

Interacción extra con DDBB

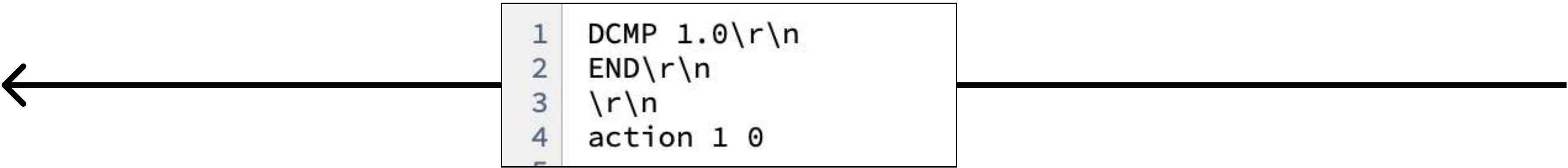
- Crear plantilla ddbb
- Aplicar migraciones

ADMP

automated drone flight management protocol



...



register code:

AAANNNNN-RRR-PPPP

AAA = COUNTRY CODE in ISO3166-1 alpha-3
NNNNN = NUMBER
RRR = REGISTRANT CODE
PPPP = DRONE PROPUSE
0001 Emergency drone
0002 Police drone
0003 Firefighters drone
...
9999 Public drone

commands:

set <instruction> <value>
action <movement_code> <meters>

movement code:

1xx:
100 Normal up
101 Slow up (!)
102 Fast up (!)
...
2xx:
200 Normal down
...
3xx:
300 Normal left
...
4xx:
400 Normal right
...
5xx:
500 Normal front
...
6xx:
600 Normal back
...
7xx:
700 Stop

code status:

1xx:
100 Server Overload
...
2xx:
200 Ok
201 Ok, but modify
...
3xx:
300 Server moved
...
4xx:
400 Syntax error
401 Authentication failed
...
5xx:
500 Server collapsed
501 Server cannot response

message type:

REQUEST -> RESPONSE
WAKE -> WAKE (Connection)
PUSH -> RECEIVE (Data transmtion)
END -> END (Route finished)
CLOSE -> CLOSE (Force close connection)

2.

Algoritmo de dirección de drones

Información que tenemos

- GPS Actual del drone
- Ruta del drone (Mapa de coordenadas, Array)
- Altitud del drone
- Orientación del drone

Minimo de 1/4 segundo para dar aviso de giro

Ejemplo:

GPS Drone: (39.482527, -0.346545)
Orientación: NE (45°)
Siguiente Punto: (39.482499, -0.346507)

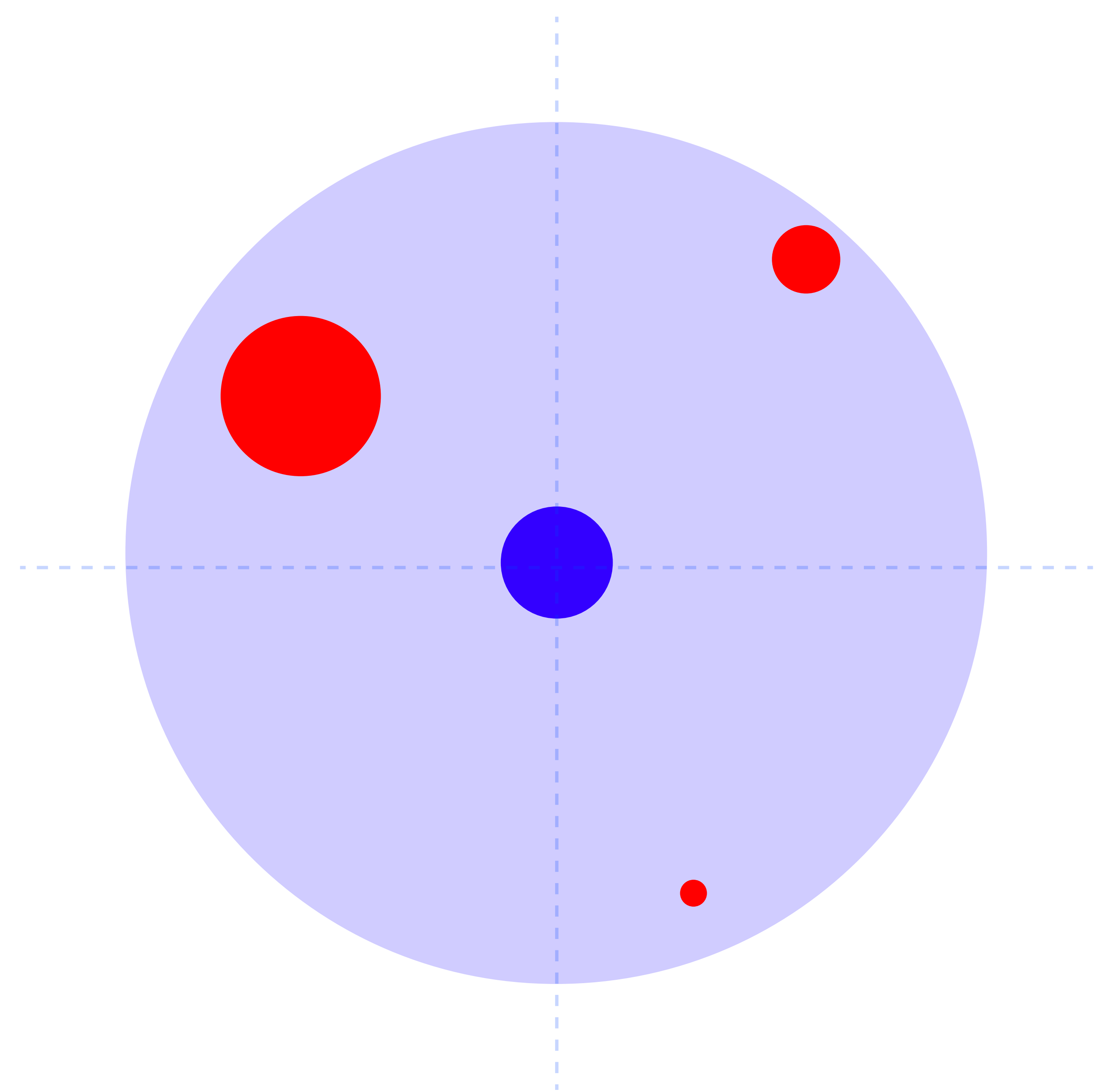
Diferencia: $\text{SiguientePunto} - \text{Actual}$
 $(-0.000028, 0.000038)$

$$\arctan(-0.000028/0.000038) = -36^\circ$$

Grado - Orientación = Grado giro
Negativo = Derecha
Positivo = Izquierda

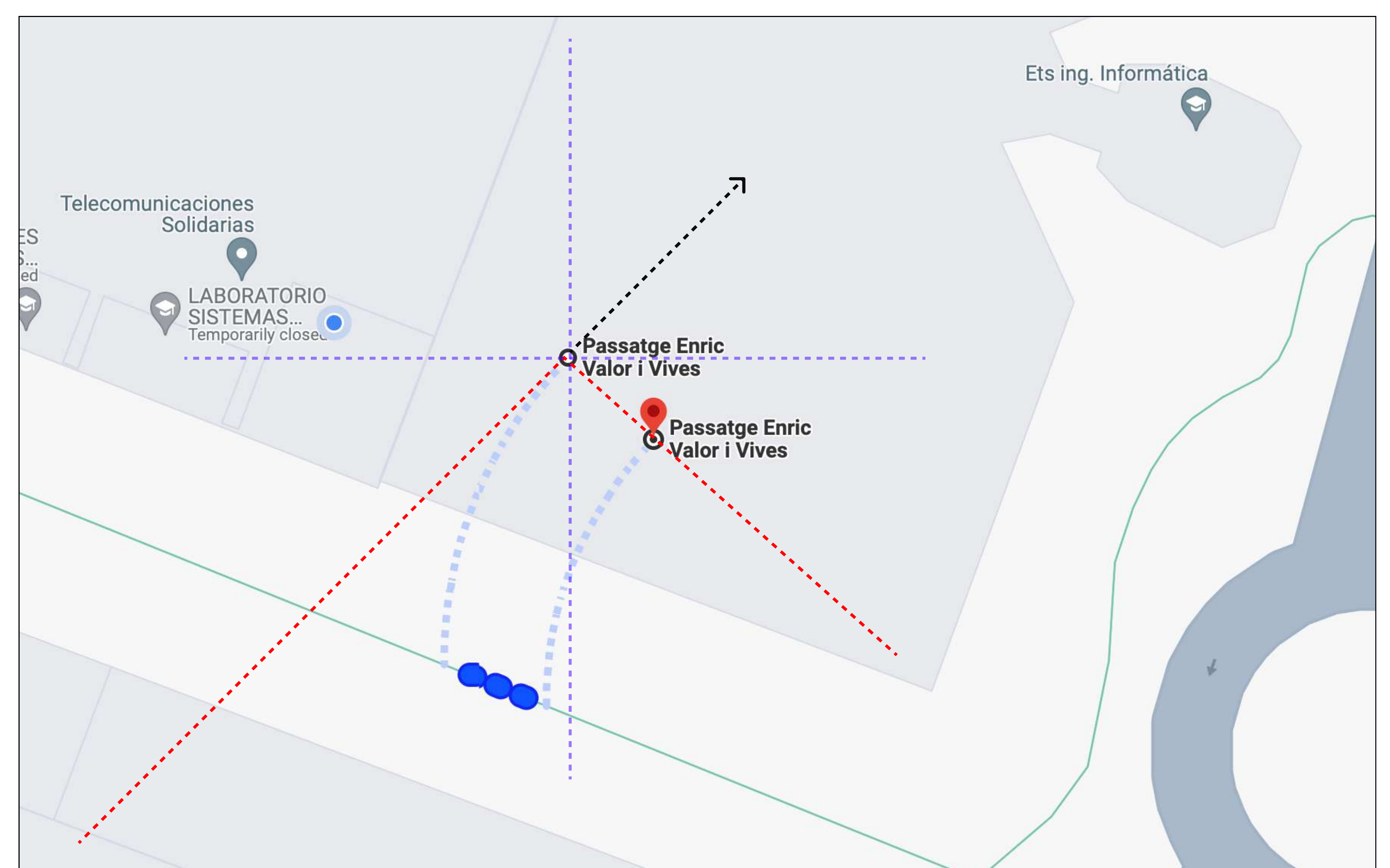
Resultado: $-81^\circ = 81$ Grados a la derecha

Algoritmo de control de choque



Se debería calcular en un radio de 100 metros los drones que están dentro. Acciones:

- * Si el dron es el más prioritario en el area
Mantener velocidad
- * Si el dron es el menos prioritario en el area
Bajar la velocidad hasta nuevo calculo
- * Si el dron es entre varias prioridades
Reducir la velocidad respecto a la escala de prioridad



Paradoja de las alturas y prioridades

Interacción con DDBB

- Guardar metadatos
- Obtener coordenadas
- Obtener alarma

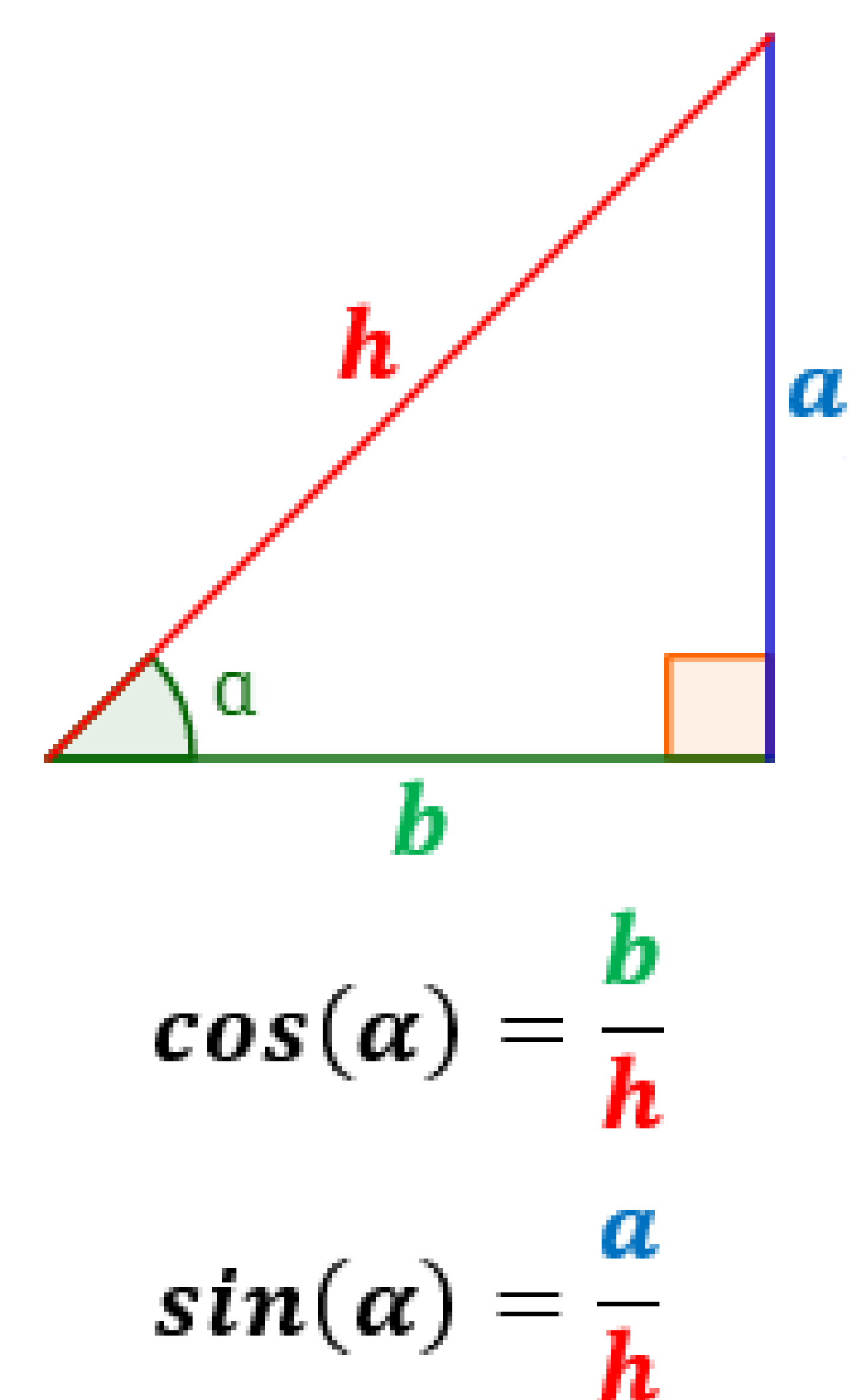
Extra info

- Cada socket atiende un drone hasta fin de route

Urgencias y predeterminado

```
En caso de error en vez de decidir
por voluntad propia. Se debe enviar
un mensaje de respuesta de tipo ERROR
con action de STOP TEMPORAL
```

Si se obtiene alarma, se pide al dron
hacer escaneo de perifericos para confirmar



3.

Algoritmo de aviso

El servidor recibe una imagen cada x tiempo (live-stream), el servidor a mandar la define y lo comunica el manager.



Modelo pre-entrenada



Código de ejemplo a seguir

Interacción con DDBB

- Guardar la alarm (true, false), alarm_code y alarm_data

```
{  
  evidence_path: "https://holamundo.es/data/10jajan18ahu1.jpg",  
  expire_datetime: "2024-03-08 21:00:00"  
}
```