RAPPORT DE STAGE:
Projet de surveillance de robots connectés à distance

Sommaire:

REMARQUE:

Les adresses IP (et les chemins d'accès) des différentes VM AWS sont à changer dans les applications en relançant les services AWS ou en les transférant vers un autre compte.

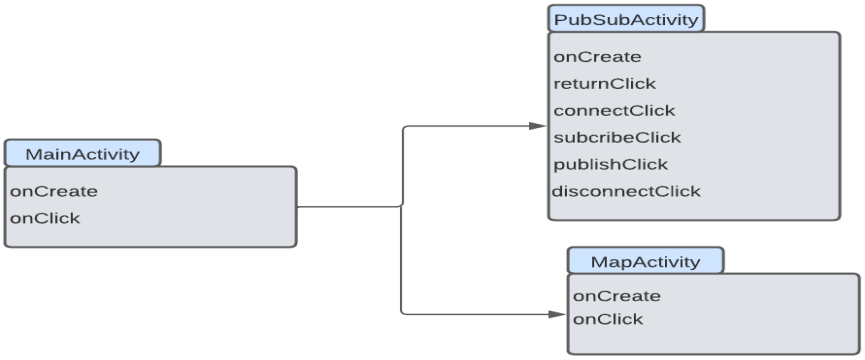
1.Application Android: VroumVroum

a. Présentation de l'application

L'application Android 'Vroum Vroum' est composé principalement de 2 parties
une partie ayant pour but de commander le robot en passant par AWS IoT Core.
une partie pour la localiser le robot et afficher sa position sur une carte.Cette partie n'a pas été implémentée car à cause de problèmes rencontrés avec la version de gradle, je n'ai pas réussi à avoir une activité fonctionnelle. De ce fait, elle ne sera pas détaillée dans ce rapport.

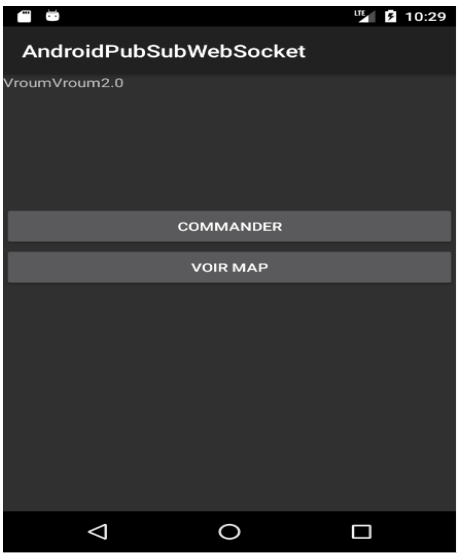
b. Architecture de l'application

Le diagramme ci-dessous présente l'architecture de l'application. Les différentes activités ainsi que les méthodes implémentées dans chaque activité.
Une activité correspond à un écran de l'application. Comme le montre la figure ci-dessus, nous avons 3 activités:



MainActivity

C'est l'écran qui sera affiché au lancement de l'application. Il nous donne la possibilité de choisir la fonctionnalité. Soit commander le robot en cliquant sur le bouton "COMMANDER", soit localiser le robot en cliquant sur le bouton "VOIR MAP".



PubSubActivity

Cette activité permet de commander le robot à travers AWS IoT Core.Pour ce faire, nous utilisons le protocole MQTT qui va permettre la communication entre le robot et l'application fonctionnant avec le principe de Subscribe\Publish.
Dans un premier temps, il faut préciser l'endpoint du compte AWS utilisé en changeant la variable *CUSTOMER_SPECIFIC_ENDPOINT* la région en changeant la variable *MY_REGION* ainsi que la variable *COGNITO_POOL_ID* .
Ensuite, une fois l'application lancée et la fonctionnalité "COMMANDER" sélectionnée, nous commençons par par connecter l'application AWS au compte client AWS qui va recevoir les commandes en cliquant sur le bouton "CONNECTER".Une fois la connexion établie, il y aura un changement dans la variable status de "Disconnected" à "connected".
Nous utilisons le topic robot/num_robot/move avec num_robot le robot qu'on voudrait commander dans le cas ou on aura plusieurs robots.Dans notre cas, ayant qu'un seul robot, on écrit comme topic robot/1/move.
En précisant le topic dans le textview à côté du bouton subscribe et en cliquant sur subscribe , ça nous permet de recevoir la dernière commande reçue sur ce topic.

En précisant le topic et la commande à envoyer dans message et en cliquant sur publish,ça nous permet de publier la commande sur ce topic dans aws iot core.Cette commande sera ensuite reçue par la raspberry relié au robot.
Les commandes à envoyer pour commander le robot sont :
F : pour faire avancer le robot
S: pour arrêter le robot
R: pour faire tourner la tête du robot à droite
L: pour faire tourner la tête du robot à gauche

MapActivity (Non implémentée)

c. Migration

Pour la prochaine personne qui développera l'application et qui compte travailler avec un autre compte AWS , il sera nécessaire de faire la migration avant de lancer l'application .Pour ce faire:

partie aws:

- Créez dans le compte aws concerné un cognito pool,sélectionnez l'option: Enable access to unauthenticated identities.

Une fois la pool créée, Cognito configure deux rôles similaires à : `POOL_NAMEAuth_Role` et `POOL_NameUnauth_Role`.

- allez dans IAM -> Rôles et sélectionnez le rôle `POOL_NameUnauth_Role`.

Nous allons maintenant attacher à ce rôle une stratégie qui dispose des autorisations pour accéder aux API AWS IoT requises.

- Appuyez sur le bouton "Attach Policy" dans l'onglet "Permissions".
- Recherchez "iot" et cochez la case à côté de la stratégie nommée "AWSIoTFullAccess", puis appuyez sur le bouton "Attach Policy".
- créez un objet
- changez la politique de l'objet en allant à l'onglet certificats
- sélectionnez le certificat dans l'onglet politiques sélectionnez la politique et modifier la version active en rajoutant une déclaration

partie android studio:

- changez CUSTOMER_SPECIFIC_ENDPOINT, COGNITO_POOL_ID et MY_REGION

partie raspeberry:

- changez endpoint et chemin des clés et certif sur le code de la carte Raspberry

2. Application: Détection de vulnérables (Projet Long)

Pour pouvoir gérer les collisions entre les différents vulnérables et robots, nous utilisons plusieurs technologies en parallèle. Un serveur distant qui reçoit en temps réel les différentes coordonnées des appareils, ainsi que les capteurs de ces dits différents appareils.

a. Serveur EC2

Le serveur distant EC2 est une machine virtuelle linux accessible via ssh en disposant d'une clé .pem qui convient. Cette machine virtuelle contient le programme Java du serveur distant qui calcule les potentielles collisions entre les appareils. La commande windows pour se connecter à la machine est la suivante :

ssh -i C:\cheminVersClePEM\cle.pem ubuntu@ec2-52-90-139-199.compute-1.amazonaws.com.

Le dernier paramètre correspond à la machine virtuelle en place sur Amazon Web Service. Pour plus de commodités, vous pouvez coder en local sur vos propres machines, mais vous devrez alors transférer les fichiers java modifiés sur la machine distante. Pour cela, il existe une commande windows tel qui suit :

scp -i C:\cheminVersClePEM\Dodo.pem C:\cheminVersCodeJava\code.java ubuntu@ec2-107-21-68-3.compute-1.amazonaws.com:/cheminVersDossierJava/dossierJava

Cela va transférer le code.java dans le dossier dossierJava sur la machine EC2.
L'architecture de dossier présente sur la machine EC2 se présente de la manière suivante :

- serveurGPS
  - ServeurGPS.jar
  - certificates
    - 95aceecb9305c301b68d46aa6b16865db0ac90cb5fcf87a4253542d15040bcb0-certificate.pem.crt
    - 95aceecb9305c301b68d46aa6b16865db0ac90cb5fcf87a4253542d15040bcb0-private.pem.key
    - 95aceecb9305c301b68d46aa6b16865db0ac90cb5fcf87a4253542d15040bcb0-public.pem.key
    - AmazonRootCA1.pem
    - AmazonRootCA3.pem
  - exec
  - lib
    - bcpkix-jdk18on-171.jar
    - ejml-fdense-0.41.jar
    - bcprov-jdk18on-171.jar
    - ejml-simple-0.41.jar

        - ejml-cdense-0.41.jar
        - ejml-zdense-0.41.jar
        - ejml-core-0.41.jar
        - gson-2.9.0.jar
        - ejml-ddense-0.41.jar
        - json-simple-1.1.jar
        - ejml-dsparse-0.41.jar
          org.eclipse.paho.client.mqttv3-1.2.5.jar
        - ejml-experimental-0.41.jar
    - manifest.txt
    - src
        - serveur
            -
            - CoordonneesGps.java
            - Kalman.java
            - MqttManager.java
            - Serveur.java
            - Tuple.java

Lorsque vous voulez ajouter des nouvelles librairies, il faut ajouter leurs jar dans le dossier lib, ainsi que de mettre à jour les fichiers manifest.txt et exec. Pour mettre à jour le code il suffit d'écraser les fichier .java situés dans le dossier src/serveur. Pour pouvoir lancer le code, il suffit ensuite d'exécuter le fichier exec (./exec dans le dossier serveurGPS) qui contient les commandes qui vont compiler et exécuter le code java. Une fois cela fait, vous pourrez alors observer le serveur en cours d'exécution.

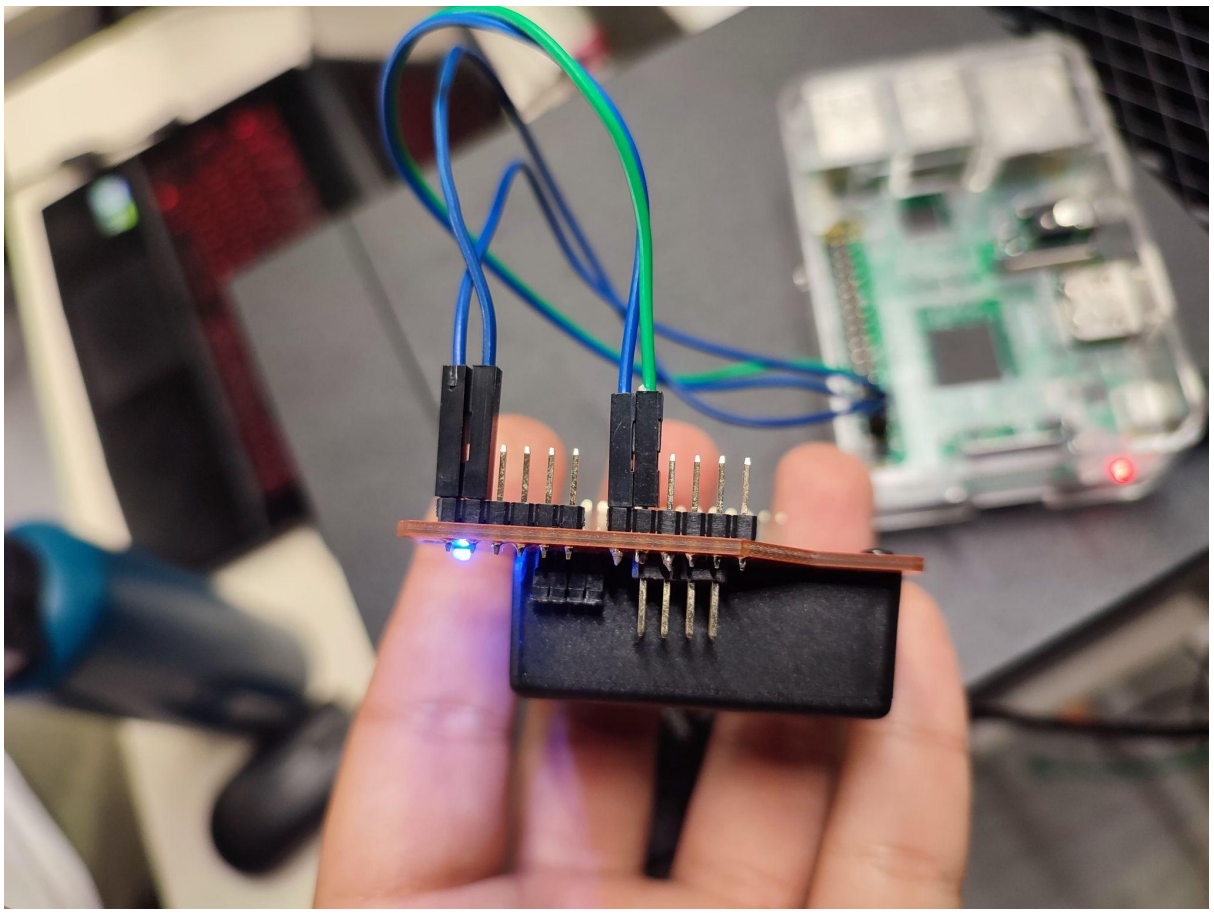
b. Application mobile GPS

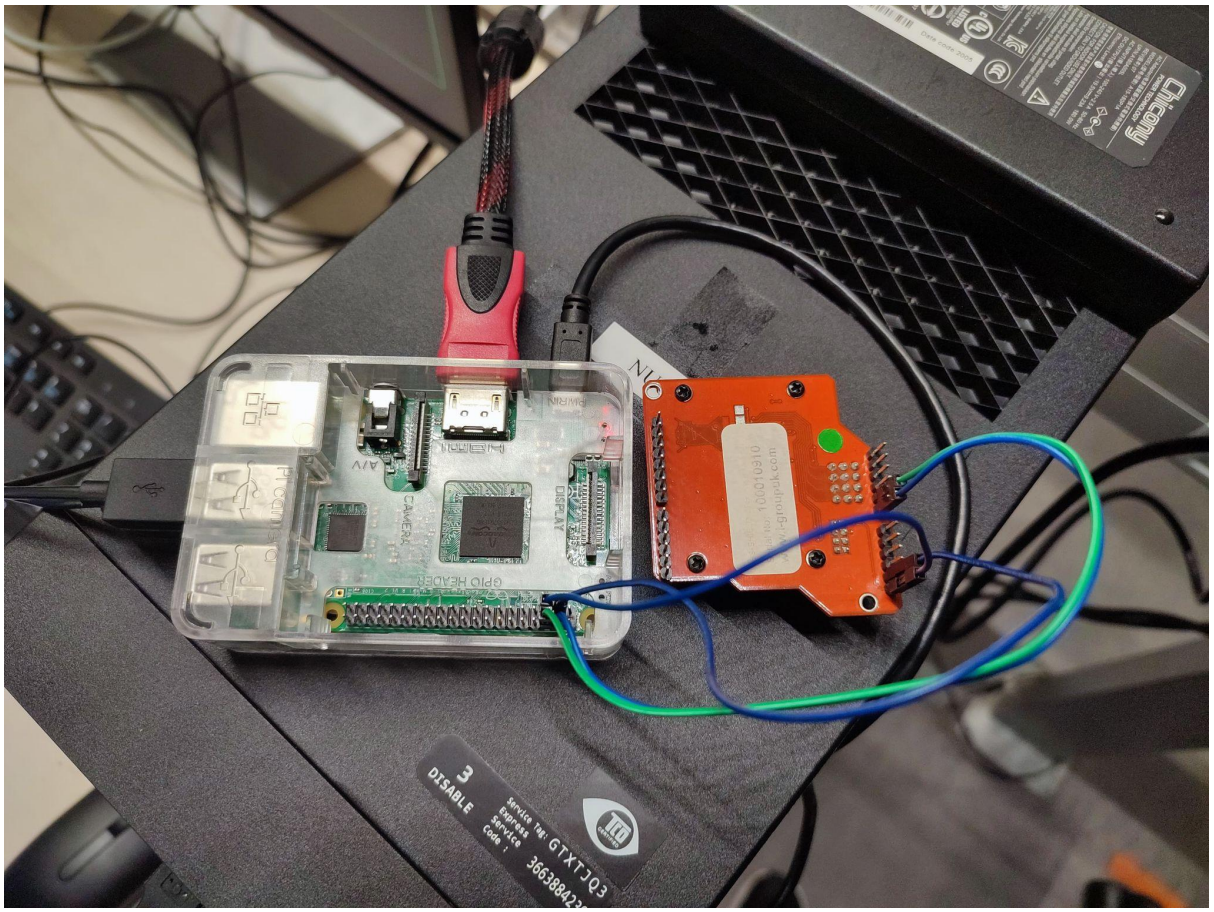
L'application mobile est une application qui sera déployée sur tous les vulnérables que l'on veut identifier dans notre environnement (piéton, vélo, etc..). C'est une application développé en Kotlin sur Android Studio

c. Application Capteur Robot

VNC Viewer est une application très pratique qui permet de visualiser l'interface graphique de la raspberry sur son propre laptop.Toutefois son utilisation est optionnelle car vous pouvez la brancher sur un écran fixe via un câble HDMI. Si vous souhaitez utiliser VNC, il faut télécharger l'application ainsi que de créer un compte sur leur site. Une fois cela fait, il faut brancher un câble ethernet à entre la raspberry et le laptop puis se connecter avec l'adresse 169.254.183.237.

Le capteur du robot est directement branché à la Raspberry Pi. Le branchement se fait selon les images suivantes :

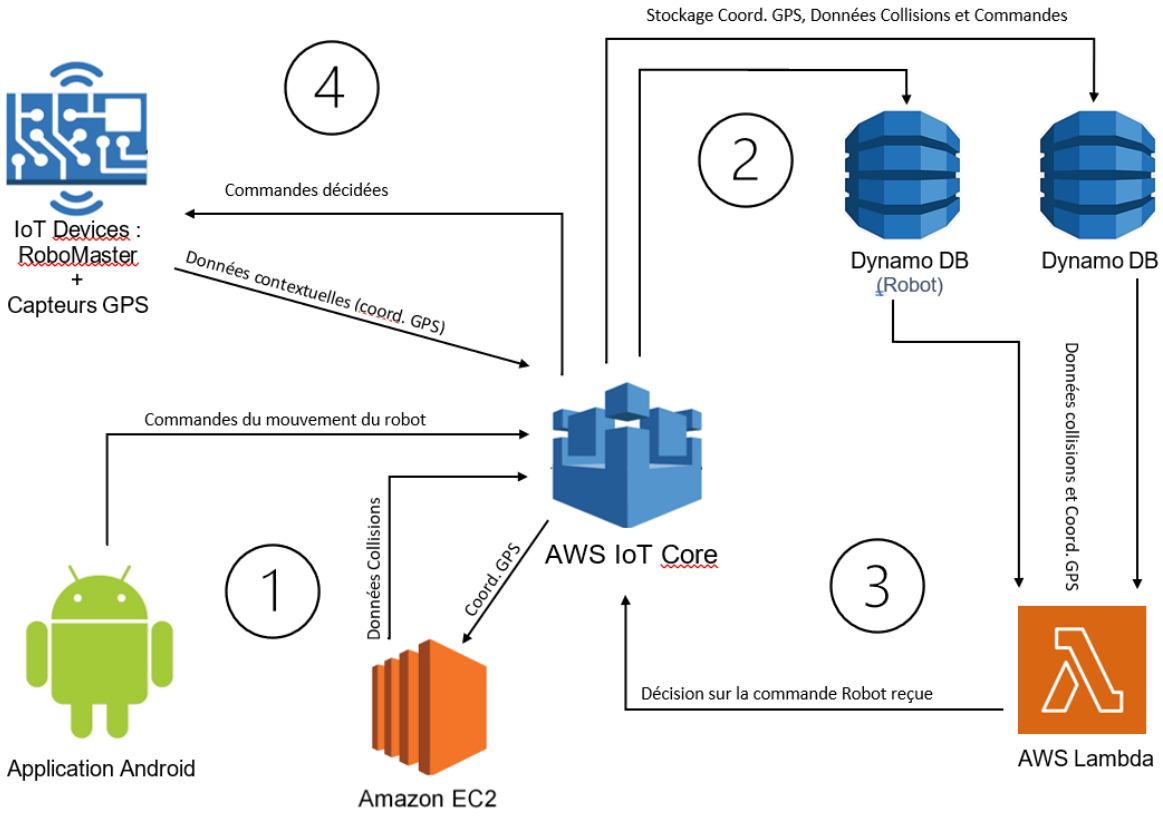



GPIO HEADER
CAMERA
DISPLAY
DISABLE
Service Tag: GTXTJQ3
Express Service Code :

Une fois le branchement effectué, rendez-vous dans le dossier contenant les codes Python pour lancer les applications.

3.Pipeline AWS

a. Vue d'ensemble



Cette architecture a été conçue dans le but de permettre et faciliter la collection, le formatage et le traitement efficaces des données échangées entre l'environnement du robot et les différentes applications.
On distingue:

- L'application Android 'Vroum Vroum': Application qui permet de commander le robot.
- L'application 'Vulnérable' (exécutée sur une machi*ne* ***virtuelle Amazon EC2***): Fait le calcul des trajectoires et la prévision des collisions.
- L'application mobile du projet long: récupère les coordonnées GPS et les renvoie à l'application 'Vulnérable'

Remarque: durant ce stage les coordonnées GPS sont récupérées d'une appli android connecté en socket au serveur EC2

b. AWS IoT Core

AWS IoT constitue le cœur de l'architecture; il permet de faire le lien entre les objets IoT de l'environnement du robot.
AWS IoT Core, en particulier, offre une série de fonctionnalités adaptées aux appareils IOT.

Nous avons utilisé AWS IoT Core pour établir une connexion (via le protocole MQTT) avec chacun des appareils connectés et leur permettre d'échanger des messages.

Ensuite, nous avons mis en place des canaux qui réagissent sur des actions bien spécifiques. Ces canaux sont créés à partir de règles de routage.
Principe de fonctionnement:

- Le canal scrute tous les topics reçus sur AWS IoT et réagit à la condition donnée en requête SQL.
- Une fois la condition vérifiée, le canal exécute une action prédéfinie.

Parmi ces canaux nous citons:

- **MQTT_to_DynamoDB collisions:** Il permet stockage données collisions sur DB en écoutant le topic 'collisions'
- **MQTT_to_DynamoDB_GPS_Robot:** Il permet stockage données collisions sur DB en écoutant le topic 'collisions'
- **iot_invoke lambda:** Il permet de lancer l'exécution de la fonction Lambda à la réception d'une nouvelle commande.

c. Fonction Lambda

Au niveau de la fonction Lambda, l'idée est de déclencher un traitement sur réception d'un ordre de déplacement du robot pour contrôler que celui-ci ne va pas entraîner de collision avant de le transférer. Si tel était le cas, le déplacement du robot serait stoppé tant qu'il y a risque de collision (avec un vulnérable ou un autre robot).

N.B.: le calcul de collision est géré séparément dans le serveur EC2.

Nous avons ici choisi une fonction Lambda comme technologie car cela nous permet de nous affranchir de la partie gestion de serveurs, tout en garantissant une efficacité plus que suffisante car elle permet de n'utiliser des ressources que lorsqu'on en a besoin. En effet, Lambda permet de définir des "triggers", qui vont déclencher l'appel à la fonction, ce qui va nous intéresser ici car on peut définir un topic d'AWS IoT comme trigger pour cette fonction Lambda. On peut ensuite effectuer le traitement permettant de vérifier qu'il n'y a pas de risque de collision à

cet instant donné et transférer la commande, ou bloquer le déplacement du robot le cas échéant.

Cependant, dans cette implémentation, s'est posé le problème de la récupération des risques de collisions, calculés depuis le serveur EC2. En effet, n'étant déclenchée que sur le topic IoT "move" associé au robot qui reçoit la commande, la fonction Lambda n'a pas accès à cette information directement. Nous avons donc utilisé une base de données DynamoDB permettant de faire tampon et de stocker ces informations pour quand la fonction Lambda en aurait besoin.

Le fonctionnement de la fonction Lambda est donc le suivant :

- sur réception d'une commande on appelle la fonction
- on récupères les informations utiles (commande, collisions)
- on regarde si le robot en question est impliqué dans une collision ou non
- on valide ou invalide la commande envoyée

d. Gestion des Bases de données(DynamoDB):

AWS offre différents services de BDD avec des spécificités différentes. Nous avons choisi pour sa modularité (possibilité de stockage de différents enregistrements et l'ajout de nouveaux attributs)
Dans le service *DynamoDB*, toute base de donnée est identifiée par une **clé primaire** (**Clé de partition**) et par sa **Clé de tri**, optionnelle.

NB: En présence de clé de tri, le couple (clé de partition, clé de tri) constitue la clé primaire de la table

Dans le cadre de ce projet:

- Clé de partition ←→ device_id
- Clé de tri ←→ timestamp (l'heure au format Epoch Unix)

MQTT_to_DynamoDB_collisions et **MQTT_to_DynamoDB_GPS_Robot** sont les deux cannaux (instanciés dans AWS IoT) qui permettent de remplir ces deux bases de données par les flux de données correspondants.

4. Axes d’amélioration

- utilisation de l’SDK d’AWS pour créer les services sous forme d’un script (donne plus de portabilité au projet)
- Améliorer l’algorithme de détection des vulnérables (changer la manière avec laquelle on calcule les collisions, Kalman)
- Afficher les coordonnées GPS des autres vulnérable quand ils sont proches (seuil à définir)
- Faire un pavé directionnel pour commander le robot au lieu d’écrire les commandes
- Finir la partie localisation du robot dans une carte

- Problème de réaction face au vulnérable: la fonction lambda qui gère la réaction du robot face à un vulnérable ne s’active que lors d’une réception d’une commande de l’application VroumVroum.
  Une solution envisageable est de créer une boucle infinie qui retransmet de façon périodique la dernière commande reçue jusqu’à la réception de la commande qui suit. (nécessite l’activation des deux canaux de ‘republish MQTT’).
  ATTENTION: l’activation de ces canaux devient vite coûteuse car dépasse rapidement le seuil du nombre de msg MQTT gratuit pendant un mois.