Travail de fin d’études

Analyse et développement d’une application mobile pour le contrôle d’un robot de cartographie 2D.

Classes utilitaires

Plusieurs classes utilitaires ont été développées pour faciliter le fonctionnement de l’application, ces classe utilitaires sont au nombre de trois.

1. MathUtils.cs

Cette classe MathUtils contient les outils nécessaires aux différents calculs effectués pour traiter les données reçues depuis l’arduino.

* Additions des distances reçues pour recréer avec précision les distances effectivement mesurées par le capteur.
* A partir de ces distances, et partant d’un angle de 0° vers 180° calculer la position des points qui serviront à dessiner le tracé, ces points sont calculés par trigonométrie connaissant les dimensions exactes de l’écran sur lequel le tracé sera affiché.
* Création d’un objet Path à partir de ces points.
* Calculer les valeurs minimale, maximale et moyenne des distances.
* Calcul des distances filtrées selon l’intervalle choisis par l’utilisateur [3,13] % 2 == 1

Ce filtrage par lissage implique de supprimer les valeurs situées au début et a la fin du tableau des distances, en effet cette moyenne lissée est calculée selon un certain intervalle, pour les distances en position : intervalle > position > distances.size – intervalle.

En effet ces distances se situant aux positions extrêmes dans le tableau, il n’est pas possible d’en calculée la moyenne lissée, le tableau des distances résultant d’un filtrage est donc plus petit que le tableau original, le tracé affiché est donc plus étroit, les angles les plus petits et les plus élevés n’étant pas calculés.

« Insérer image »

1. Codes.cs

La classe Codes.cs est une classe statique utilisée pour accéder facilement aux commandes à envoyées vers le robot ces commandes sont stockées sous un format prêt à être envoyé sur le réseau.

1. PaintManager.cs

La classe PaintManager.cs contient les outils graphiques de l’application, des objets de la classe Paint, ceux-ci sont utilisés pour donner une couleur, une forme pour dessiner l’écran d’affichage des tracés.

Architecture de l’application mobile

Architecture Model-View-Presenter

L’architecture choisie pour l’application mobile est une architecture MVP ( Model-View-Presenter) , cette approche aide à découpler l’affichage le model et la vue.

Les vues utilisées sont les Activités, native de Android , celles-ci sont utilisées pour l’affichage de l’application ainsi que l’accès à certaines fonctionnalités propres à Android , telle que les accès aux composants hardware.

Les presenters quant à eux sont des classes qui tiennent le rôle de contrôleurs, mais contrairement aux contrôleurs du model MVC, les presenters ne se contente pas de mettre les données à jours selon la logique choisie, les évènements de l’interface graphique sont directement transférés vers les presenters et ces derniers ont la responsabilité de répondre à ces évènements.

Le model quant à lui est très simple pour cette application, il ne comprend qu’une classe qui conserve les tracés mesurés par le robot.

Pour permettre de découpler au maximum l’activité du presenter, tout deux implémente une interface « IActivity » et « Ipresenter » , le presenter à un attribut de type IActivity tandis que l’activity a un attribut IPresenter ce qui permet de limiter fortement le couplage entre affichage et contrôle.

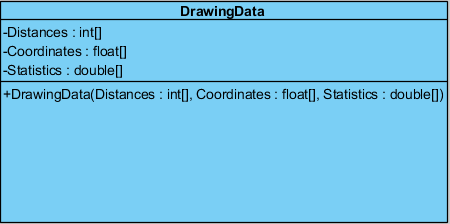
Cette disposition améliore la flexibilité et la testabilité du code.

En contre-partie, l’architecture MVP à le désavantage d’augmenter légèrement la quantité de code

MVP dans cette application.

1. Model

Pour cette application le modèle est composé d’une seule classe, la classe « DrawingData » qui contient toutes les données relatives à un balayage effectué par le robot.



Les instances de cette classe sont créées lorsque le robot renvoie un tableau des distances mesurées.

A partie des mesures reçues, la classe MathUtils est utilisée pour calculer les attributs du nouvel objet DrawingData.

1. Vues

Les vues sont affichées sur des fichiers « Activity », ceux-ci sont des dispositions natives d’Android constitués de deux fichiers, un fichier de code Activity.cs et un fichier de description xaml, les composants graphiques peuvent être placé facilement via le fichier xaml(en mode design ou source) mais peuvent aussi être placé de manière programmatique via le fichier Acitivity.cs, cette façons de faire étant plus longue et non nécessaire dans mon cas, tous les composants graphiques sont ajoutés via le fichier descriptif xaml.

Ecran d’accueil

Fichiers : MainActivity.cs, IMainActivity.cs, main.xaml

C’est l’écran sur lequel s’ouvre l’application

« Intégrer écran accueil annoté»

1. Entrer une adresse ip vers laquelle se connecter.
2. Entrer un port de connexion.
3. Restaurer ces deux paramètres par défaut.
4. Ouvrir l’écran de sélection.
5. Ouvrir l’écran de contrôle.

Ecran de contrôle

L’écran de contrôle est l’écran principal de l’application, à partir de cet écran, selon qu’il soit ouvert en mode pilote ou en mode review il permet de diriger le robot, d’éffectuer un balayage, de filtrer les données, les enregistrer ou les supprimer.

« Intégrer écran controle review »

« Intégrer écran contrôle guidage »

Ecran listage des tracés

Cet écran est composé d’une liste d’éléments cliquables

Pour ce faire chaque élément contient le nom exact du fichier à ouvrir, lors du click ce nom est concaténé au répertoire des tracés et envoyé dans l’écran de contrôle en mode review ou ce tracé est alors affiché.

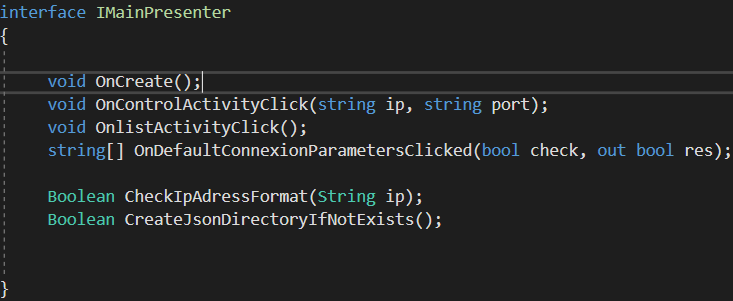
« Intégrer écran liste »

1. Presenters

Les presenters sont au nombre de 3, un par vue (Activity), ces presenters ont plus qu’un simple rôle de contrôleur, ils reçoivent directement les évènements des interfaces graphiques et sont chargé d’y répondre adéquatement.

1. MainPresenter

Le MainPresenter implémente une interface IMainPresenter :



Le MainPresenter est chargé de la gestion des évènements de l’activité MainActivity, chaque évènement de click sur cette activité est transféré au MainPresenter.

Le rôle de ce presenter est de réagir lorsque l’utilisateur ouvre la l’écran de contrôle (OnControlActivityClick) et à l’ouverture de l’écran de listage (OnListActivityClick).

Il est aussi utilisé pour initialisé le répertoire où seront stockés les tracés, la vérification du format de l’adresse ip fournie ainsi que la réinitialisation des paramètres de connexion par défaut.

1. ControlPresenter
2. ListPresenter

Communication Android-Arduino

La communication entre l’appareil mobile et le robot se fait via en wifi, la carte arduino yun initialisant un point d’accès propre auquel l’application mobile vient se connecter à la demande.

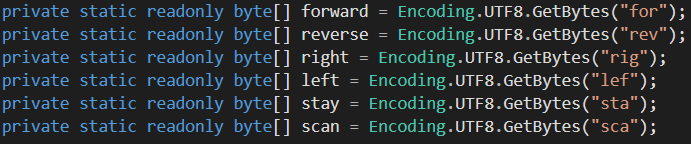
L’utilisateur doit d’abord entrer les paramètres de connexion sur la page d’accueil, ils comprennent une adresse ip et un port, ces deux paramètres sont déjà complétés par défaut mais peuvent être modifier.

En cliquant sur le bouton Commande, l’utilisateur ouvre l’écran de contrôle du robot, l’application tente de se connecter en utilisant les paramètres fournis à l’écran précédent, en cas de succès l’écran de contrôle s’ouvre en mode « Contrôle », en cas d’échec de la connexion, l’application revient à l’écran d’accueil en affichant un message d’erreur à l’utilisateur.

Du côté Android, les commandes sont stockées sous la forme de bytes array, seul type pouvant être porté par la connexion, ces commandes sont peu nombreuses :

* marche avant
* marche arrière
* gauche
* droite
* rester
* scan
* connecter
* deconnecter

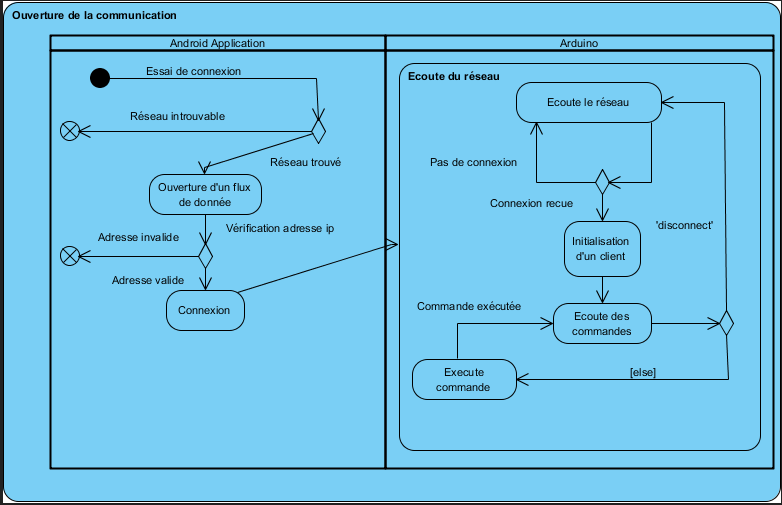
Elles sont stockées sous la forme :



Du côté arduino, un programme tourne en boucle en attente d’un client, avant cela, lors de l’allumage de la carte un server (« BridgeServer » ) est initialisé, celui-ci écoute par défaut le port 5678.

Le programme arduino écoute donc en boucle l’arrivé d’une connexion, si c’est le cas, un objet client est créé et le programme entre dans une nouvelle boucle, il s’agit maintenant d’écouter les commandes envoyées par l’application mobile, l’écoute se poursuit jusqu’à l’envoie de la commande « disconnect », à partir de ce moment le programme retourne à la boucle de départ, en attente d’une nouvelle connexion.

Les requêtes reçues par l’arduino sont converties en String et comparées aux différentes commandes reconnues, si la requête est correcte alors le programme dirige le robot comme souhaité et continue a écouter de nouvelles commandes.



Guidage du robot

Une fois ouvert en mode « Control », l’écran « CommandActivity » permet entre-autre de déplacer le robot, pour ce faire, on fait appel a l’accéléromètre de la tablette/smartphone, ce senseur est disponible sur la majorité des appareils mobiles du commerce.

Lorsque l’utilisateur presse le bouton « Pilot » , le senseur accéléromètre est « enregistré » cela signifie qu’il débute la détection des mouvements de la tablette selon les 3 axes X,Y,Z mais seul les valeurs des axes X et Y nous intéresse car le robot ne se déplace qu’en deux dimensions.

L’énumération (Enum) SensorDelay propose différents intervalles possible entre deux mesures , « Fastest », « Game », « UI », « Normal » , c’est l’intervalle Normal qui a été choisis, il propose une mesure chaque 200.000 microsecondes soit 5 mesures par secondes, il n’est pas nécessaire de mettre à jours les valeurs plus souvent, en effet le robot se déplace lentement.

Lorsque le senseur est enregistré il débute donc la mesure des mouvements de la tablettes chaque 1/5 seconde, et transmet via la méthode OnSensorChanged(SensorEvent e) les valeurs X et Y au ControlPresenter , celui se charge de transmettre la commande correspondante vers le robot soit

X <= -2 : Marche avant

X > 2 : Marche arrière

Y <= -2 A gauche

Y > 2 A droite

Arrêt si aucune des conditions n’est atteinte.

Lorsqu’e l’utilisateur relâche son doigt, le senseur est désenregistré et n’effectue plus de mesure jusqu’à la prochaine utilisation du bouton.

Scanner

1. Envoi de la commande

En mode "Control" l'utilisateur peut lancer un scan de l'environnement avant du robot, sur un angle de 180 degrés.

En pressant le bouton "SCAN" il envoi la commande correspondante au micro-controleur.

Cette commande est stockée sous forme de byte array :



1. Calcul et réception des données

Une fois la commande reçue par le robot, on entre alors dans un processus de calcul des distances, le capteur tourne sur 180 degrés et effectue une mesure de distance par degré d'angle et effectue un second calcul sur la trajectoire de retour, les données sont enregistrées dans deux tableaux et sont ensuite additionnées deux par deux et le résultat de chaque addition est divisé par deux pour obtenir une moyenne et limiter une première fois les valeurs aberrantes.

Une fois ce scan terminé, un nouveau tableau est créé, ce tableau contient 360 valeurs et non 180 car le type byte ne supporte que des valeurs maximales de 255 (centimètres) or nous voulons récupérer et afficher des valeurs jusqu'à 400 centimètres, pour ce faire chaque valeur est décomposées en deux, une première partie avec 255 pour maximum et une seconde avec 145 pour maximum, ces valeurs seront plus tard sommées deux par deux dans l'application mobile pour retrouver la distance originale.

Une fois le nouveau tableau prêt, celui-ci est envoyé à l'application mobile qui crée un objet "Drawing" à partir des distances reçues, celles-ci étant ordonnées de 0 à 179 degrés.

L’utilisation du scan est bloquante pour le reste de l’application, c’est un choix délibéré, l’utilisateur ne pouvant bien sur pas déplacer le robot lorsque celui-ci effectue une mesure

Pour créer cet objet, les distances reçues sont additionnées deux à deux pour récupérer la valeur initiale, une fois fait, les valeurs minimale, maximale ainsi que la moyenne sont calculées.

1. Traitement
2. Enregistrement

Les tracés ne sont pas enregistrés sous forme d’image, seules les données permettant d’afficher le dessin sont conservées dans un objet « insérer », cet objet contient toutes les données nécessaires pour réafficher le dessin par la suite.

Ces objets sont sérialisés au format JSON en grâce à une librairie externe appelée Newtonsoft.JSON , les données à sérialisées sont uniquement constituées de types numériques qui sont facilement utilisables en JSON, ce format est lisible aussi bien pour la machine que pour l’humain, contrairement à une sérialisation binaire, de plus il est très simple de l’utiliser pour partager des données vers d’autres applications, la même librairie est également utilisée pour la désérialisation de l’objet lors d’un futur affichage.

1. Consultation

L'utilisateur peut facilement consulter les tracés, ceux si sont accessibles à partir de l’écran d’accueil en cliquant sur le bouton « liste » ou ouvre une nouvelle interface « DataListActivity » , il suffit de cliquer sur l’un des tracés affichés pour ouvrir celui-ci dans l’interface de contrôle, celle-ci sera alors ouverte en mode « Review » permettant uniquement la consultation du tracé, le filtrage et l’enregistrement/suppression d’un tracé, il n’est pas possible de contrôler le robot dans ce mode

« Insérer écran d’accueil avec focus sur le bouton»

« Insérer écran DataListActivity»

« Insérer écran Control review»

1. Suppression