1 Introduction

1.1 Thème . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.2 Pourquoi ce thème . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.3 Environnement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.3.1 Matériel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.3.2 Logiciel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.4 Un peu de théorie . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.4.1 Arduino . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2 Analyse

2.1 Arduino . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2 Android . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3 Développement

3.1 Arduino . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.2 Android . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.2.1 Activités . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.2.2 Connexion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4 Méthodes

4.1 Pilotage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2 Détecter les obstacles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.1 Scanner . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.2 Tracer les obstacles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Introduction

Le sujet de mon travail de fin d’études est le développement, d’une application mobile permettant le pilotage d'un robot, celui-ci contrôlé par une carte Arduino. Cette application Android permet de guider le robot dans un espace donné et lui demander de détecter les obstacles situes sur un angle de 180 degrés. Une fois les informations récupérées du robot, l’application permet également de tracer un dessin su‑segment précis de l'environnement faisant face au robot. Ce dessin peut être adapté en fonction des valeurs choisies et sauvegardé par la suite dans une liste que l’application propose.

Pour la structure, ce travail contient 3 grands chapitres couvrant une partie introductive, d’explication de l’environnement choisi (hardware et software) ainsi que quelques concepts théoriques que j’ai rencontré sur le parcours, une deuxième partie regroupant l’analyse et le développement de l’application mobile et enfin un dernier chapitre destiné à la conclusion ainsi qu’une dernière partie expliquant les contraintes auxquelles ce sujet a été soumis, ainsi que les pistes d’amélioration envisagées.

Concernant la programmation, pour ce travail j’ai décidé de privilégier le développement de l’application mobile, en C# via Xamarin. La programmation du microcontrôleur Arduino, avec son propre langage venant d’une adaptation du C++, va être intégrée, elle, dans cette analyse de l’application.

Ce thème m'a paru être une bonne occasion pour mettre en pratique plusieurs facettes du développement informatique, regroupant à la fois la programmation embarquée, la programmation mobile ainsi qu'une partie de gestion de la communication entre les deux composantes. Ce projet m’a permis de me former à ces différents aspects que je n'ai pas encore pu découvrir précédemment. Un challenge ambitieux donc qui m’a permis d’envisager plusieurs alternatives pour des problèmes diverses, de software et de hardware également.

Chapitre 1 : Environnement et Concepts Théoriques

* 1. Le Matériel Hardware

En dehors du développement lui-même, ce projet a nécessité l’achat d’un robot avec l’objectif se déplacer dans un espace donné, de pouvoir capter les informations de l’environnement extérieur, les enregistrer et les envoyer à l’application mobile pouvoir les traiter.

Pour ce faire, le robot doit contenir quelques éléments importants : le microcontrôleur, le capteur et le châssis.

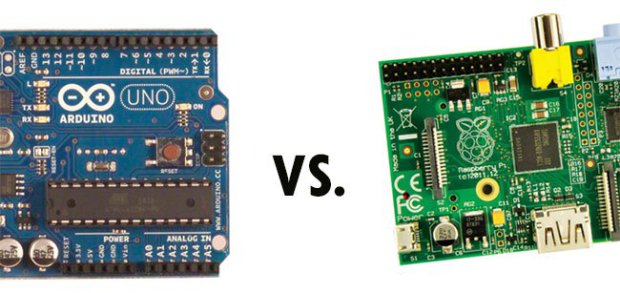
Le Microcontrôleur

« Un microcontrôleur est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte et mémoire vive), unités périphériques et interfaces d'entrées-sorties. (…) Les microcontrôleurs sont fréquemment utilisés dans les systèmes embarqués, comme les contrôleurs des moteurs automobiles, les télécommandes, les appareils de bureau, l'électroménager, les jouets, la téléphonie mobile, etc. »[[1]](#footnote-1).

« Le microcontrôleur est composé de quatre parties : un microprocesseur (…) qui va exécuter le programme embarqué dans le microcontrôleur, une mémoire de données (RAM ou EEPROM) (…), une mémoire programmable (ROM), qui va contenir les instructions du programme pilotant l'application à laquelle le microcontrôleur est dédié (…). La dernière partie correspond aux ressources auxiliaires. Celles-ci sont généralement : ports d'entrées / sorties parallèle et série, des timers pour générer ou mesurer des signaux avec une grande précision temporelle et des convertisseurs A/N pour traiter les signaux. »[[2]](#footnote-2)

Ce genre de micro-ordinateurs, on en retrouve partout, des portes du garages qui sont contrôlées à distance jusqu’aux lave-linges qui sont programmées pour s’arrêter à une certaine heure, tout fonctionne à l’aide de ces mini-ordinateurs. Dans mon cas, c’est le microcontrôleur qui dirige le robot. Il lui permet de changer de position, doit diriger le robot et lui permettre de changer de position, avancer, reculer, capter les informations, les enregistrer et les envoyer, par la suite, à l’application mobile. Il communique, de la sorte avec l’application mobile.

Le choix du microcontrôleur a donc été une étape importante de mon projet. uite à une analyse des différents produits existants sur le marché, deux alternatives m’ont paru les plus pertinentes : l’Arduino ou la Raspberry Pi.



A la différence d’un Arduino, qui est un microcontrôleur plutôt simple à utiliser puisqu’il exécute un seul programme à la fois, la Raspberry Pi est « un ordinateur à usage général, généralement avec un système d'exploitation Linux, qui a la possibilité d'exécuter plusieurs programmes en même temps. »[[3]](#footnote-3) . Ce dernier, tout en étant beaucoup plus performant, s’adresse surtout à des robots plus complexes, utilisant plusieurs programmes différents en même temps. Il est donc ; a priori, plus difficile à utiliser qu’un Arduino. C’est pourquoi mon choix c’est porté sur une *« Arduino Yun mini* ».

*L’Arduino Yun mini* est composée de deux éléments : le microcontrôleur Arduino classique qui utilise un processeur ATmega32u4, ainsi qu’un second processeur Atheros AR9331. Le premier est utilisé pour la gestion des ports GPIO (General Purpose Input/Output) que j'appellerais « broches » pour la suite. C'est aussi sur ce processeur qu'est chargé le sketch (programme) mis en œuvre par la carte. Il est donc responsable de la communication avec les autres composants du dispositif.

Le second supporte une distribution Linux (Linino Os) basée sur OpenWRT, ce qui permet à *l'Arduino Yun* d'être utilisée également comme un micro-ordinateur. Grâce à la communication entre les deux processeurs qui s'effectue via une librairie « Bridge », il est ainsi possible d'exécuter des scripts Shell ou Python et d'avoir une interaction avec le programme chargé dans l'Arduino. La carte dispose également d'un port USB-A, Ethernet et WIFI intégré.

Arduino est difficilement capable de gérer à elle seule le fonctionnement des deux moteurs. Dans ce but, un double pont H (L298N) est branché pour simplifier la commande des moteurs.

Le capteur

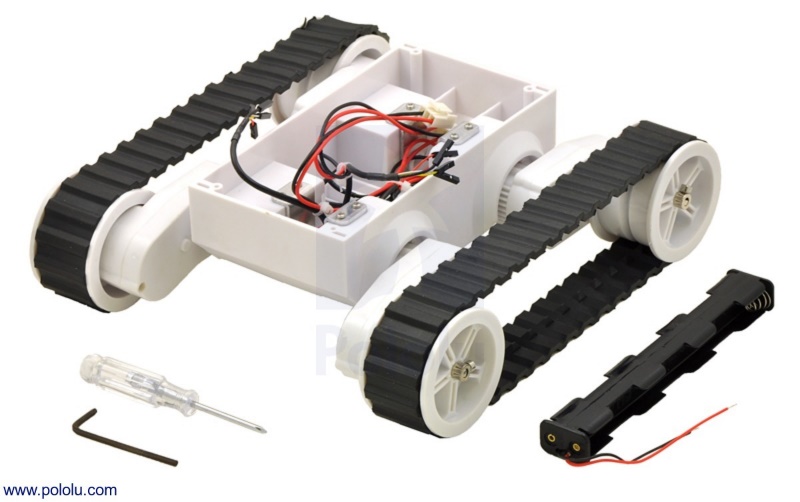
Pour arriver à dessiner une tracée de ce qui se trouve devant le robot, un capteur est essentiel, faute de quoi l’information extérieure ne peut pas être captée. Pour le choix du capteur, plusieurs choix ont été également possibles : le laser, les infrarouges, l’ultrason, etc…

Les infrarouges sont très dépendantes de la couleur, des surfaces, des textures, mais aussi des angles et de la taille de la pièce. Le noir, par exemple, absorbe la lumière, donc les mesures sont moins précises, alors que le blanc reflète la lumière envoyant des données qui ne sont pas conformes. Ce type de capteur n’est donc pas assez performant pour mon projet.

L’ultrason est également sensible à l’environnement, même si dans une moindre mesure. C’est pourquoi, idéalement le capteur laser est souhaitable pour une qualité optimale des données recueils. Pour des raisons économiques, cependant, j’ai été contraint de me limiter à l’ultrason. Mon choix s’est donc porté sur le « HC-SR04 »[[4]](#footnote-4).

En principe, le fonctionnement du capteur ultrason est comme suit : le microcontrôleur envoie du courant dans la broche « trig » du capteur. En réaction, celui-ci émet une série d’ondes à 40khz qui se réfléchissent sur un objet et reviennent vers le capteur. On peut ensuite calculer la distance de cet objet en fonction de la durée de déplacement de l’onde. Le capteur tourne sur 180° par pas de 1° pour chaque angle. On peut donc créer, à la suite de ce processus, une carte de l’environnement faisant face au robot.

Le Châssis

Enfin, le châssis choisi pour permettre le déplacement du robot est le « Dagu Rover 5 ». Le châssis comprend un support de batterie et deux moteurs à courant continu, avec un entraînement indépendant et un encodeur en quadrature pour chaque bande de roulement. Le mécanisme permet également de configurer le jeu du châssis en ajustant l'angle des ensembles de boîte de vitesses. C’est donc un châssis chenillé (pour faciliter le déplacement) complet auquel on doit simplement ajouter le contrôleur et les différents capteurs pour que tout fonctionne dans les meilleures conditions.

1.2. Le Matériel Software

Le développement de l'application mobile est réalisé à l'aide de l'environnement de développement Visual Studio 2017 et Xamarin pour le support d'Android.

« Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# utilisent tous le même environnement de développement intégré (IDE), qui leur permet de partager des outils et facilite la création de solutions faisant appel à plusieurs langages. »[[5]](#footnote-5)

« Xamarin est une plate-forme de développement d'applications mobiles permettant de créer des applications natives iOS, Android et Windows à partir d'un code C # / .NET commun, réalisant 75% à près de 100% de réutilisation du code entre les plates-formes. Les applications écrites avec Xamarin et C # ont un accès complet aux plates-formes API sous-jacentes et la possibilité de créer des interfaces utilisateur natives et de compiler des packages spécifiques à la plate-forme. »[[6]](#footnote-6) Le choix s’est porté sur Xamarin puisque cela permet un développement en C# avec lequel j’étais habitué.

Pour l’Arduino la programmation est réalisée via Arduino Ide, éditeur permettant de compiler le code et le charger directement dans la mémoire de l'Arduino.

Pour information, un programme Arduino se nomme 'Sketch'. C'est donc ce terme que j'utiliserai par la suite dans mon analyse. Un sketch Arduino commence toujours par une méthode Setup(). Celle-ci est appelée au démarrage du programme et n'est exécutée qu'une seule fois, suivant chaque démarrage ou reset. On l'utilise uniquement pour l'initialisation de variable ou pour donner le sens des broches (INPUT/OUTPUT).

Dans ce travail elle est utilisée pour donner le sens des broches de moteur, servo et capteur ainsi que pour initialiser un serveur qui écoutera le réseau. Cette méthode est obligatoire dans tout programme Arduino. Ensuite, une seconde méthode est utilisée, Loop(), comme son nom l'indique, il s'agit d'une méthode qui s'exécute en boucle, c'est le cœur du sketch et elle donne sa réactivité au programme. La communication entre les deux processeurs de l'Arduino est facilitée par la librairie 'Bridge'. Toutes les commandes passées via cette classe sont interprétées par Python du côté du processeur Atheros AR9331 (Linux). Elle sert entre-autre à la communication réseau ou encore à l'exécution de script sous Linux.

Chapitre 2 Analyse 2.1 Arduino L e microcontroleur arduino et l'application mobile sont reliés via une connexion wi établie depuis arduino en mode Acces Point, l'addresse ip xe donnée à l'arduino est toujours là même : 192.168.240.1 mais il est possible de la modier depuis l'interface web arduino, le port d'écoute est déni sur 5678. Pour la gestion logicielle des deux moteurs, nous avons besoins de 3 pin's par moteur, 2 servant à donner le sens de rotation du moteur et un troisième pour indiquer la vitesse de rotation. P our chacuns de ces pins il faut donner le mode d'utilisation INPUT/OUTPUT , dans ce cas-ci bien sur c'est le mode OUTPUT qui est utilisé. 2.2 Android P our rappel, une interface utilisateur comprend à la fois un chier 'Activité' ou éventuellement un chier 'Fragment' et un layout XMl disposant les diérents composants de la présentation. Dans notre cas, quatres interfaces existent, une interface principale, une interface de commande du robot, une interface de présentation des tracés enregistrés et une quatrième pour a‑cher un tracés sélectionné. L 'UI principale est utilisée pour indiquer les informations de connexion , l'adresse ip et le port sur lequel l'application doit essayer de se connecter, par défaut ces informations sont celles stockées dans le chier Strings.xml. Elle sert aussi de point d'entrée vers l'interface de commande et l'interface de présentation de tracés enregistrés (Boutons 'LISTE' et 'START' sur l'image suivante)

Chapitre 3 Developpement 3.1 Arduino L e développement de la partie arduino est simplié par l'utilisation du langague arduino en lieu et place de l'habituel langage C souvent utilisé pour programmer des microcontrolleurs. Il s'agit d'un langage dérivé du c++ et fournissant des methodes de haut niveau pour le contrôle des éléments hardware et simpliant grandement l'écriture du code. Le développement arduino a été la partie la plus courte en terme de programmation, l'essentiel étant la réception et l'envoi de données vers l'application mobile et le traîtement des requêtes recues, celles-ci sont au nombre de sept, elles sont transmises par wiet converties en string, elles peuvent avoir pour valeur 'connect', 'disconnect', 'forward', 'backward', 'right', 'left', 'stay' ou 'scan' et sont traîtées en fonction. Nous Devons utiliser la librairie 'Bridge' qui simplie fortement la communication entre les deux processeurs de l'arduino, les commandes passées via Bridge sont ensuite interprétées par Python du côté Linux. Deux autres librairies sont utilisées, 'BridgeServer' qui nous permet d'écouter des communications venant du réseaux, si un le serveur accepte une connection, il renvoi un object de la classe 'BridgeClient' qui nous permet d'écouter les requêtes et d'envoyer les réponses à l'application mobile

3.2 Android C haque composant de l'application (Connexion, Direction, Dessin, Enregistrement) à été codé et testé séparemment avant le développement d'une application complète et l'intégration de tous ces composants. L 'application étant réalisée exclusivement pour Android je me contenterai de parler du fonctionnement d'applications sous cet OS et pas sous IOS ni Windows Phone. Une application mobile pour Android se présente sous la forme de vues dénies au formal XML (AXMl), ces vues peuvent contenir des layout et des controles (widget) tels que boutons,liste,checkbox ou beaucoup d'autres , qui permettent d'intéraction avec l'utilisateur. Ces layout et widgets peuvent être positionnés soit par un cliquer/glisser depuis la boite à outil soit directement via le chier axml. exemple Checkbox :

D errière chaques vues xml se trouve un chier de code appelé `Activity`, cet Activity est responsable des intéractions entre composants de la vue et le modèle, dans le cas de Xamarin ces chiers ont l'extension .cs (c sharp). Lors de la création du projet, Xamarin crée automatiquement une Activité MainActivity.cs, la methode SetContentView permet de charger la vue comme contenu de l'écran.

L a première chose à faire est alors de référencer les éléments graphiques de notre vue :

L 'élément defaultConnexionParam est récupéré grâce à son ID dénit dans le chier XML, dé lors il est possible de l'utiliser dans le code, il en va de même pour les autres widgets TextView, Button, ToggleButton et tous les autres. 3.2.1 Activités L 'application est divisée en quatres `Activities` , MainActivity, DrawingActivity, ListImageActivity et ImageActivity, ces activités sont reliées à quatres vues diérentes : Main.axml, Map.axml, ListImage.axml, ImageView.axml. MainActivity se charge de récuréper l'adresse ip et le port nécessaire à la connexion avec l' arduino, une checkbox permet d'utiliser des paramètres par défaut conserver dans le chier strings.xml 3.2.2 Connexion L a connexion entre l'application android et le microcontrolleur arduino se fait par wi, l'adresse ip par défaut étant 192.168.240.1 tandis que le port choisi et hard-codé est 5678. Du côté arduino , nous dénissons un Server qui écoutera sur le port 5678

La première ligne indique simplement que notre serveur écoute les connexion venant directement du réseau, la seconde demande au serveur de commencer l'écoute. 8 E nsuite, notre methode Loop() est lancée, elle boucle jusqu'a l'arrivée d'un client :

L orsque notre serveur détecte la connection d'un client , il initialise un objet BridgeClient qui a pour rôle la communication de et vers l'application mobile, si un client est détecté alors on commence a écouter la communication tant que le message d'arret `D` n'est pas reçus. P lusieurs commandes peuvent êtres traîtées par le programme, 'forward', 'backward', 'right', 'left', 'stay' et 'scan' servent à diriger le robot et a demander la détection des obstacles.

Chapitre 4 Méthodes 4.1 Pilotage L e guidage du robot se fait depuis l'écran de dessin de carte, pour cela un bouton personalisé est utilisé, l'activité DrawingActivity implémente l'interface IOnTouchListener qui permet de capter les évènements de toucher sur l'écran ainsi que l'interface ISensorEventListener qui permet d'utiliser , si le touché a lieu sur le cercle de pilotage, alors notre activité commence à utilisé le capteur :

d é lors, tant que l'on maintient le doigt sur ce cercle les mouvement de la tablette sont capté et utilisés, le delai entre deux détection étant très court SensorDelay.Normal , il s'agit de l'interval le plus long mais il reste bien trop rapide pour guider un robot si lent, j'utilise donc une méthode pour récupérer le moment de chaque mise à jours (en nanosecondes) auquel on soustrait le temps de la dernière mise à jour utilisée,

D e cette façon, la direction n'est plus mise à jours que 4 fois par seconde, ce qui est largement su‑sant. La direction à prendre est envoyée vers le robot via wi en utilisant la classe NetworkStream. Une fois recue par l'arduino, la boucle principale transmet la commande sous forme d'une String vers la methode SetDirection, qui prend en paramètre un string de direction 'forward', 'backward', 'right' ou 'left', cette méthode est utilisée pour donner aux deux moteurs le sens de roulement correspondant, ensuite, une seconde methode 'Go()' est appelée pour démarrer les moteurs. Chaque moteurs utilise 3 pin de connexion , deux sont utilisés pour indiquer le sens de rotation et le troisième pour donner la vitesse au moteur. Lorsqu'on relâche le bouton de direction, c'est alors la commande 'stay' qui est envoyée au robot et les moteurs sont simplement mis à l'arrêt en attendant les commandes suivantes. 4.2 Détecter les obstacles 4.2.1 Scanner L e scan commence par l angle 0 degrés et se poursuit dans une boucle jusqu'a 180 degrés, pour chaque angle, on eectue un calcul de distance à l'aide du capteur ultrason, les distances sont enregistrées dans l'ordre pour chaque angle de 0 a 180 degrés. La méthode est la suivante, Un signal est envoyé à la broche d'émission pour lui signier qu'elle doit émettre une série de 8 impulsions à 40 kHz Ensuite, la broche de réception écoute si un signal lui revient, si c'est le cas, la broche de réception passe au niveau HIGH pendant le temps du trajet, on peu ensuite diviser ce temps pour calculer la distance de l'obstacle, cette étape est répétée pour chaque angle de 0 à 180 degré. Les distances sont stockées dans un tableau de bytes car ces sous ce format que les données sont transférées via wi, étant donné que la valeur maximale d'un byte de 8 bits est de 255 et sachant que notre capteur a une distance maximale de détection de 400 cm, chaque distance recue est coupée en une partie haute et une partie basse.

D e ce fait , un appel à la methode de scan remplit un tableau de 2\*180 = 360 éléments, une fois le scan terminé, ce tableau est envoyé vers l'application mobile ou les distances initiales sont restituées en additionnant simplement les valeurs 2 à 2

Analyse.

2.1 Arduino

La programmation du microcontrôleur Arduino est l’une des étapes pour la réalisation de ce travail.

Le langage utilisé est le C++ utilisant la bibliothèque Arduino, celle-ci simplifie le développement de code spécifique au microcontrôleur.

Trois classes spécifiques de cette bibliothèque sont utilisées pour le contrôle de ce robot.

1. BridgeServer

La classe BridgeServer, permet à programme arduino d’avoir un rôle de serveur et d’écouter les connexions entrantes sur un port déterminé, lorsqu’une connexion est acceptée, une instance de la classe BridgeClient est créée.

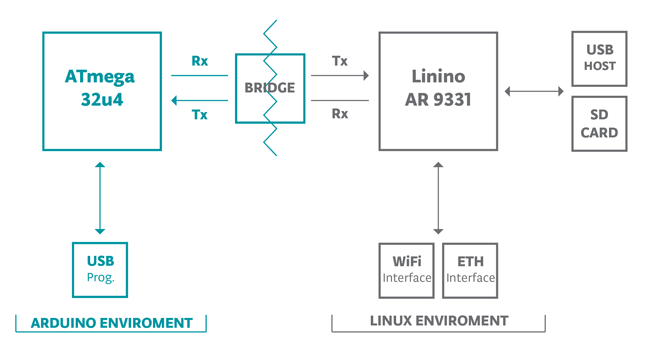
1. BridgeClient

L’instance de BridgeClient ainsi créée est utilisée pour recevoir les commandes envoyées par le terminal mobile, ainsi que pour l’envoi de données (contours d’un balayage) vers l’application.

1. Servo

Une classe dédiée au contrôle de servomoteurs, ceux-ci servent à diriger le capteur à ultrason.

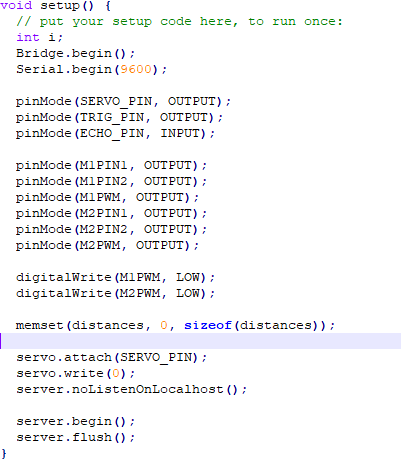
D’une manière générale, les classes « Bridge » font le lien entre les deux processeurs du microcontrôleur, le processeur Atheros AR9331 fait tourner une distribution Linux dédiée (Linino Os) à arduino, c’est ce processeur qui gère toutes les communications réseau, tandis que le second processeur Atmega 32u4 est responsable de la programmation des sketchs.

[[7]](#footnote-7) Illustration https://www.arduino.cc/en/Guide/ArduinoYun

Un sketch Arduino se compose au minimum de deux méthodes

1. Setup

Cette méthode est toujours la première exécutée lors du lancement d’un programme Arduino, comme son nom l’indique elle sert normalement à l’initialisation de variables, l’ouverture d’un serveur ou encore à initialiser les modes des pins en temps que I/O, c’est également là que l’on « attache » les pins à différents composants matériels.



1. Loop

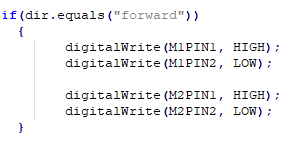
Le programme Arduino en lui-même, le flow, dans notre cas, il s’agit d’une boucle qui écoute constamment les commandes en provenance du terminal mobile et y répond adéquatement soit en déplaçant le robot, soit en l’arrêtant, soit en faisant un balayage et en envoyant ensuite les distances mesurées.



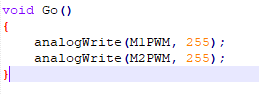
Déplacement du robot

Le déplacement du robot est réalisé grâce à deux moteurs, chaque moteur est relié à trois GPIOs du microcontrôleur, tous trois utilisés en mode output, dont deux digitaux On/Off et l’un analogique permettant d’envoyer une valeur particulière.

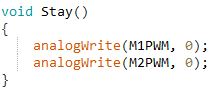
Lorsqu’une commande de déplacement est envoyée, par exemple la commande de marche avant, les pins mis en mode digital, sont placé soit HIGH, soit LOW, cette configuration permet de choisir le sens de rotation du moteur, pour la marche avant, les deux moteurs doivent tourner en sens inverse :



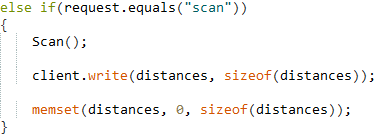
Il suffit ensuite de donner au troisième GPIO une vitesse de rotation pour les moteurs, cette vitesse a pour maximum 255.



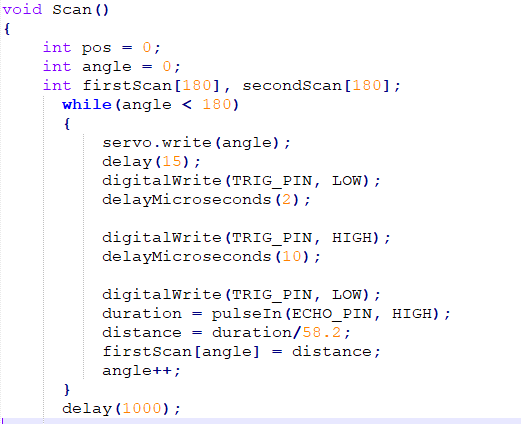
Pour arrêter le robot il suffit mettre cette valeur à 0.



Si la commande reçue est celle d’un balayage

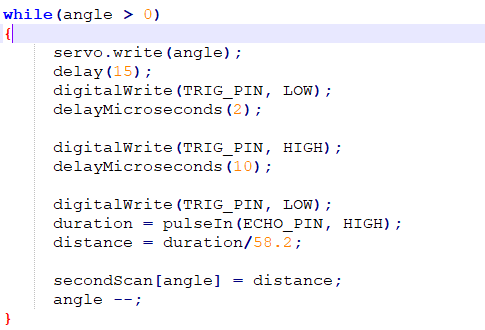


La méthode Scan est appelée :

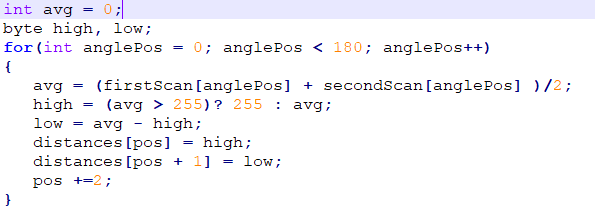


Après avoir initialisé deux tableaux d’entiers pour contenir les distances mesurées, elle fait faire un demi-tour au capteur pour mesurer une distance à chaque degré d’angle.

Un second passage en sens inverse est réalisé et une moyenne des deux mesures est prise et placée dans un tableau destiné à être envoyé vers l’application mobile.



Cette dernière partie calcul une moyenne des deux mesures et place dans un tableau deux valeur par mesure, une valeur haute et une valeur basse, pour les valeurs supérieures à 255, qui est la valeur maximale d’un byte.



2.2 Android

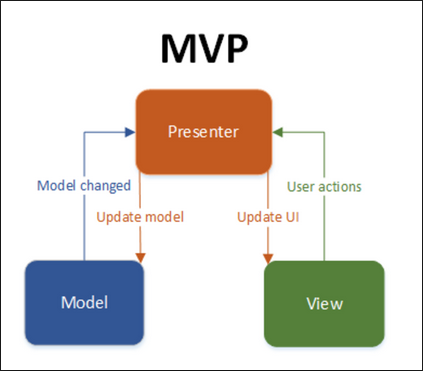
L’essentiel du travail de développement a été effectué sur la partie application mobile (Android 7.0 Nougat).

Cette application mobile a pour rôle la connexion vers le robot, le contrôle de ses différentes fonctionnalités, ainsi que la gestion des données qui résultent de ce contrôle, tel que l’affichage des données reçues, sous la forme d’un nuage de points reliés, le filtrage de ce nuage de points pour limiter les valeurs aberrantes qui apparaissent inévitablement, ainsi que l’enregistrement et la suppression de ces données.

Pour parvenir à ce résultat une application composée de trois écrans plus une fenêtre de paramètre à été développée, cette application a été réalisé en utilisant le langage C# et la technologie Xamarin, permettant de réaliser des applications mobiles en utilisant ce langage.

Architecture

L’architecture retenue pour le développement de l’application mobile utilise le model MVP, Model-View-Presenter, assez similaire au MVC, Model-View-Contrôler, elle en diffère au niveau du contrôleur (présentateur dans ce cas).



[[8]](#footnote-8)

Le model de notre application repose sur trois classes

1. Coordinates.cs (Struct)

Elle représente la partie la plus basique de notre model, une paire de coordonnées X, Y chaque objet de cette classe représente donc un point dans le plan.

Sous la forme d’un « Struct » structure de donnée plus légère qu’une classe et facilement utilisable pour un model simple et qui ne sera pas modifié après sa création.

1. Points.cs (Struct)

Cette classe représente un point dans le plan ainsi que sa distance par rapport au robot, elle a comme attribut un objet coordinates ainsi qu’un entier représentant la distance.

1. ContoursData.cs (Classe)

Cette classe contient les données liées à un balayage (détection des contours de l’environnement), elle représente les données persistantes de l’application, pour se faire elle est sérialisée au format Json pour être conservée dans le dossier de l’application

Elle comprend un tableau de Points, un tableau de statistiques et un tableau de distances.

Dans le cas d’une application Android, les composants View sont en fait composés de plusieurs fichiers différents mais liés

Un fichier de description au format xml, celui-ci sert à décrire l’aspect visuel des éléments graphiques à l’écran, dans ce fichier sont décrits, entre-autres, leur taille, leur position sur l’écran, le style donné, mais aussi parfois directement leurs réactions à des évènements utilisateur, un clic sur un bouton, un scroll ou autres.

Un fichier de code, une activité, celle-ci est liée au fichier xml, les composants graphiques peuvent être contrôlés depuis cette activité, on essaie de rendre l’activité la plus « bête » possible, passive et sans connaissance du model, elle se contente de recevoir les évènements de l’écran, et de les répercuter vers le presenter.

L’activité est également chargée de l’interaction avec les composants hardwares du terminal, dans notre cas, l’une des activités interagit avec le capteur accéléromètre du terminal.

Les presenters jouent un rôle de contrôleur élargit, ils reçoivent les entrées utilisateur qui ont été transférés par la vue, et décident en conséquence de la marche à suivre, et de la mise à jour éventuelle de la vue et/ou du modèle, ceci dans le but de rendre la vue la plus passive possible et d’empêcher la connaissance du modèle par la vue.

Outre ces trois types de composants, un quatrième dossier de fichiers est présent dans notre application, «Utilitaires» on y trouve des classes qui servent uniquement d’aide, d’utilitaire, les objets de cette classe de sont pas persistant , ne font pas partie du model et sont utilisés par les presenters pour faciliter leur tâche, calcul des coordonnées de chaque point, filtrage, calcul des nouvelles coordonnées , couleur et styles pour le tracé des contours .

Quatre classes utilitaires(helper)

1. Codes.cs

Cette classe contient des références aux codes utilisés dans l’application, tel que les commandes de contrôle envoyées au robot ou encore les id de messages pouvant être affichés dans des boîtes de dialogue.

1. MathUtils.cs

La classe contient une série de méthodes d’aide mathématique, principalement trigonométrique, permettant de calculer les coordonnées des points à afficher, de calculer une moyenne lissée ou de définir le contour d’un tracé.

1. PaintManager.cs

Cette classe contient un dictionnaire d’objet « Paint », une classe destinée à des opérations de dessin, un Paint permet de définir l’épaisseur du trait , sa couleur, le type de trait.

1. DrawingData

La « Traduction graphique » du model de tracé, un objet de cette classe est créé à partir d’une instance de la classe «ContoursData» , cette classe a pour attributs les différents composantes graphique d’un tracé.

Elle deux instances de la classe Path, décrite plus haut, l’une d’elle représente le tracé exact calculé à partir des mesures, la deuxième un tracé plus général ne passant que par une partie des points mesurés.

Une instance de cette classe est créée en utilisant un opérateur de conversion qui permet de la créer à partir d’un objet ContoursData

« Capture de ContoursData => DrawingData »

Trois écrans.

Chacun des trois écrans est le résultat de l’interaction entre trois composants majeurs : une vue GUI (Graphic User Interface, sous la forme d’un fichier axml), une activité (Activity) et un presenter (contrôleur).

Le rôle de la vue est de présenter les éléments sous forme graphique à l’utilisateur, elle se présente sous la forme d’un fichier axml descriptif, c’est depuis ce fichier que la disposition des éléments graphiques est décrite, leur taille, leur couleur, leur identifiant.

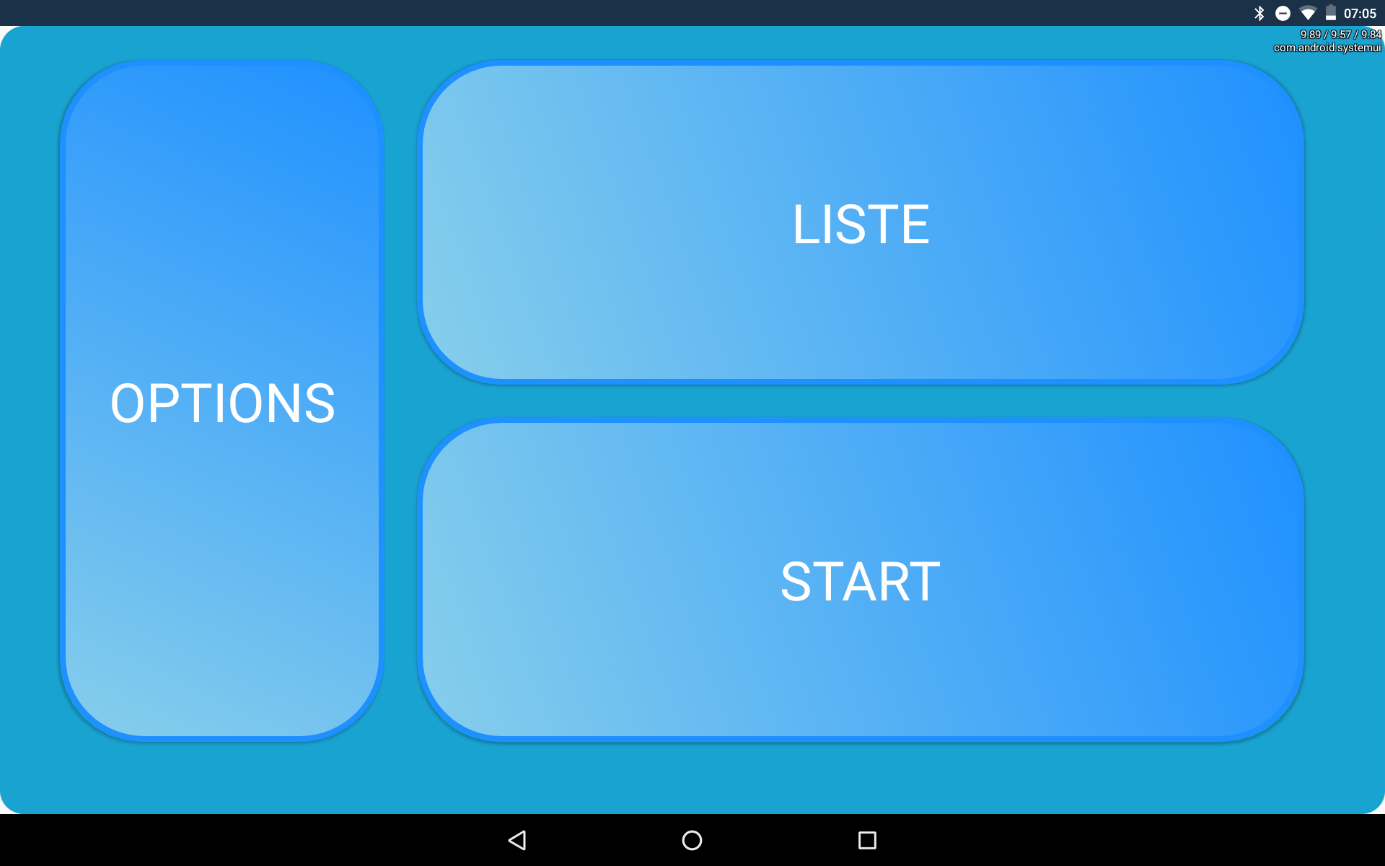
Pour interagir avec cette vue, une activité (Activity) est utilisée, c’est elle qui se charge de fournir un contexte d’affichage pour la vue et permet ainsi le contrôle de ses composants, ceux-ci sont liés « programaticalement » grâce à leur identifiant (Id) présent dans le fichier xml.

Le rôle d’une activité est également d’interagir avec les éléments hardwares du terminal, dans notre cas nous faisons appel aux senseurs du terminal, principalement son accéléromètre.

Outre ces deux éléments obligatoires, un troisième a été ajouté, selon le model MVP (Model-View-Presenter) , les réactions aux évènements pouvant avoir lieu sur l’écran, tel que le clic sur un bouton sont définies au niveau de l’activité, ces réactions font appel a des méthodes du presenter, qui décident de la suite des opérations, ce presenter tiens le rôle d’un contrôleur mais avec donc plus d’importance, il est responsable non seulement du contrôle des informations qui seront transmises au model mais il décide également des réactions à avoir en fonction des évènements sur la vue.

Le premier écran est un écran d’accueil, depuis celui-ci on peu accéder à la totalité des fonctions de l’application, trois boutons le composent, permettant respectivement

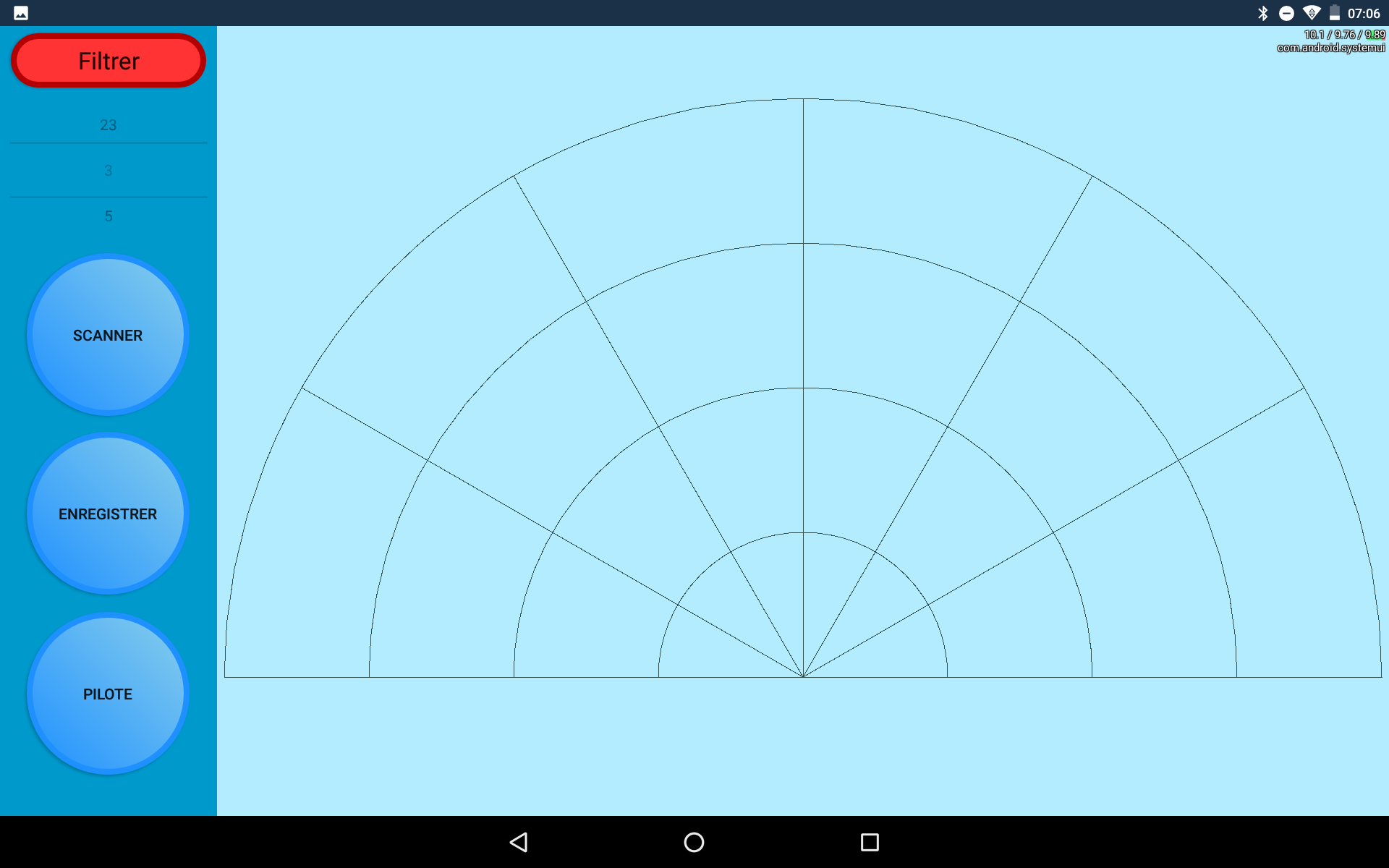
1. L’ouverture d’une fenêtre d’options de connexion.
2. L’ouverture de la liste des tracés enregistrés.
3. L’ouverture de l’écran de contrôle du robot.



Ecran de contrôle

L’écran de contrôle du robot se présente en deux parties, la partie contrôle elle-même se trouve sur la gauche, on y retrouve quatre boutons et un sélectionneur numérique.

1. Bouton Filtrer, ce bouton permet d’activer le filtrage du nuage de points.
2. Sélectionneur numérique, il permet de choisir l’intervalle utilisé pour le filtrage, ce filtrage est basé sur une moyenne lissée.
3. Bouton Scanner, il envoie la commande prévue au robot pour lancer le balayage.
4. Bouton Enregistrer, il permet d’enregistrer un tracé.
5. Bouton Piloter, en maintenant le doigt sur ce bouton, on récupère l’état du capteur accéléromètre du terminal, cet état est utilisé pour diriger le robot.



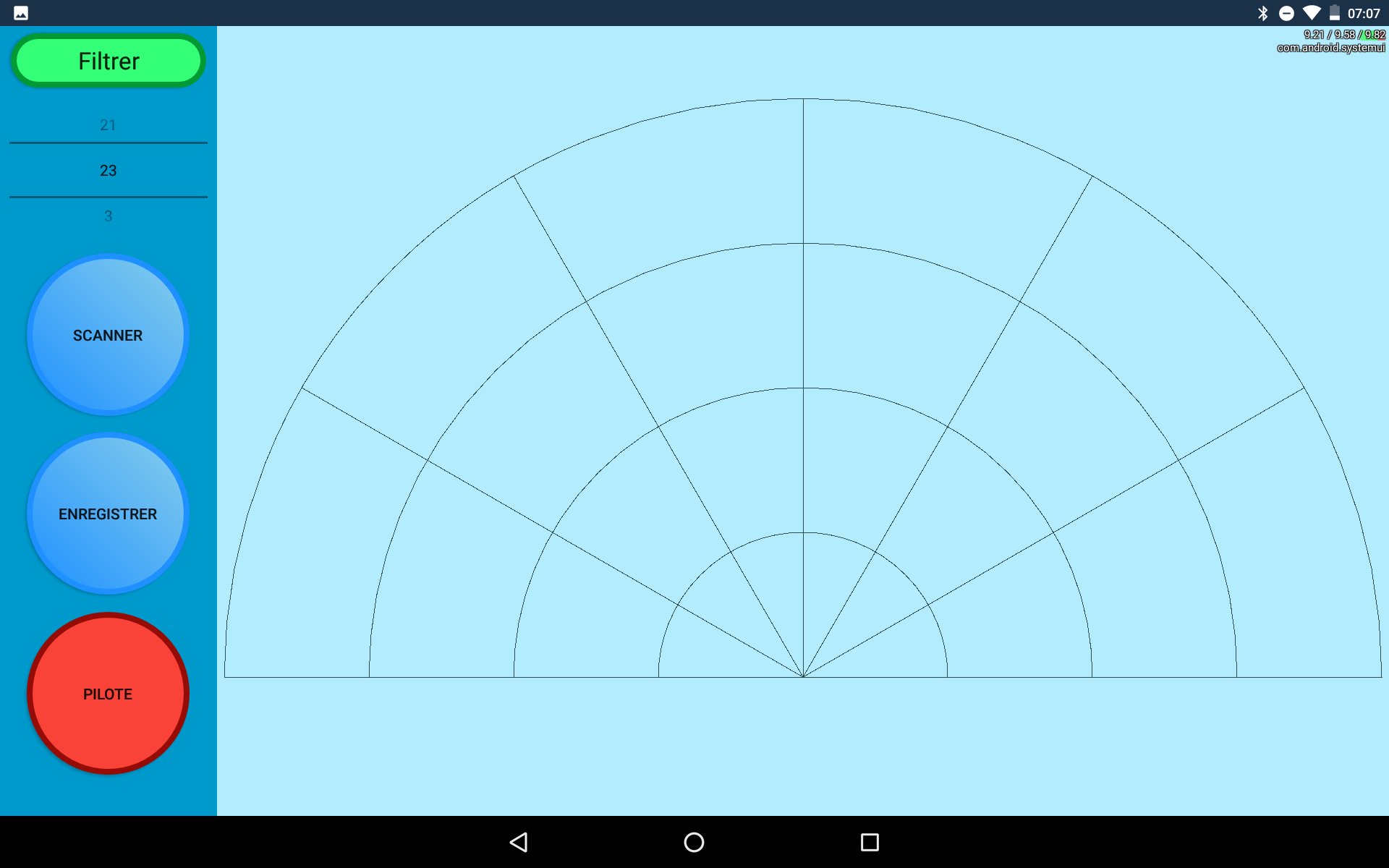
Fonctionnalités écran de contrôle

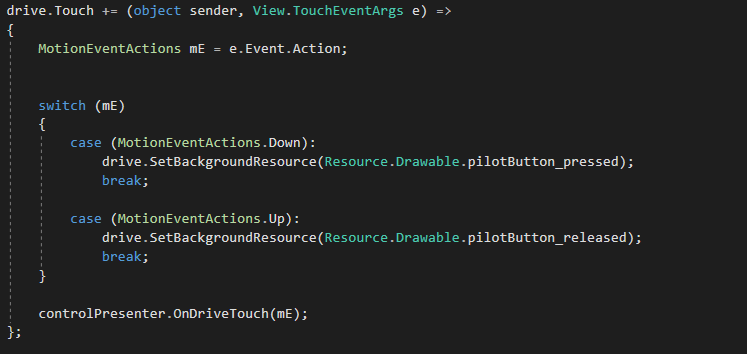
1. Diriger le robot

Le guidage du robot se fait donc comme nous l’avons vu, à partir de l’écran de contrôle, pour commencer à déplacer le robot il faut maintenir le doigt sur le bouton Piloter et pencher le terminal dans la direction souhaitée.

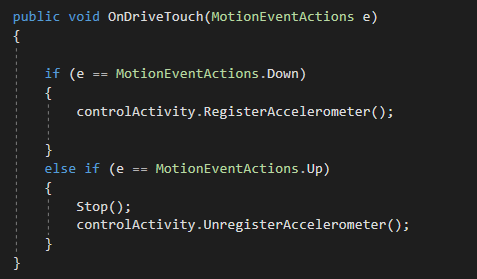
Le fonctionnement est plutôt simple, lorsqu’e l’utilisateur touche le bouton de pilotage, l’activité de contrôle enregistre le capteur accéléromètre du terminal, cela signifie que l’activité reçoit désormais les informations envoyées par ce capteur.

L’activité détecte un toucher sur le bouton de pilotage et répond en conséquence : modifier l’apparence du bouton de pilotage et transmettre l’action vers le presenter :

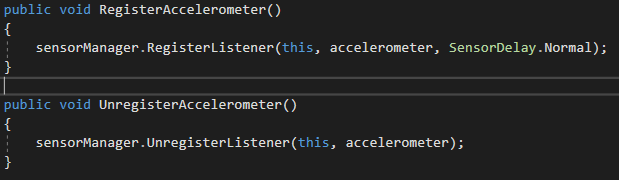




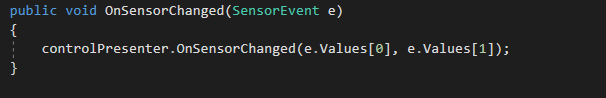
Le presenter reçoit cette action et y réagit en conséquence, demandant à l’activité d’enregistrer le capteur ou au contraire de le désenregistrer :



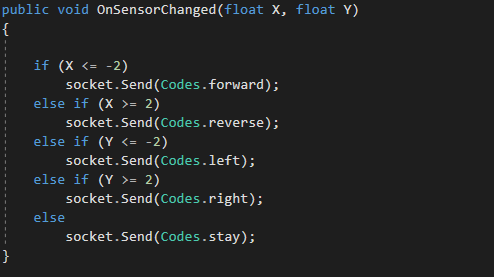
Du côté de l’activité, le capteur est donc soit enregistré utilisant un délai régulier entre deux mesures, définit par « SensorDelayNormal »



Lorsqu’e le capteur est enregistré, l’activité capte donc les mouvements du terminal et envoie le résultat au presenter sous la forme de deux valeurs, accélération sur l’axe x et accélération sur l’axe y :



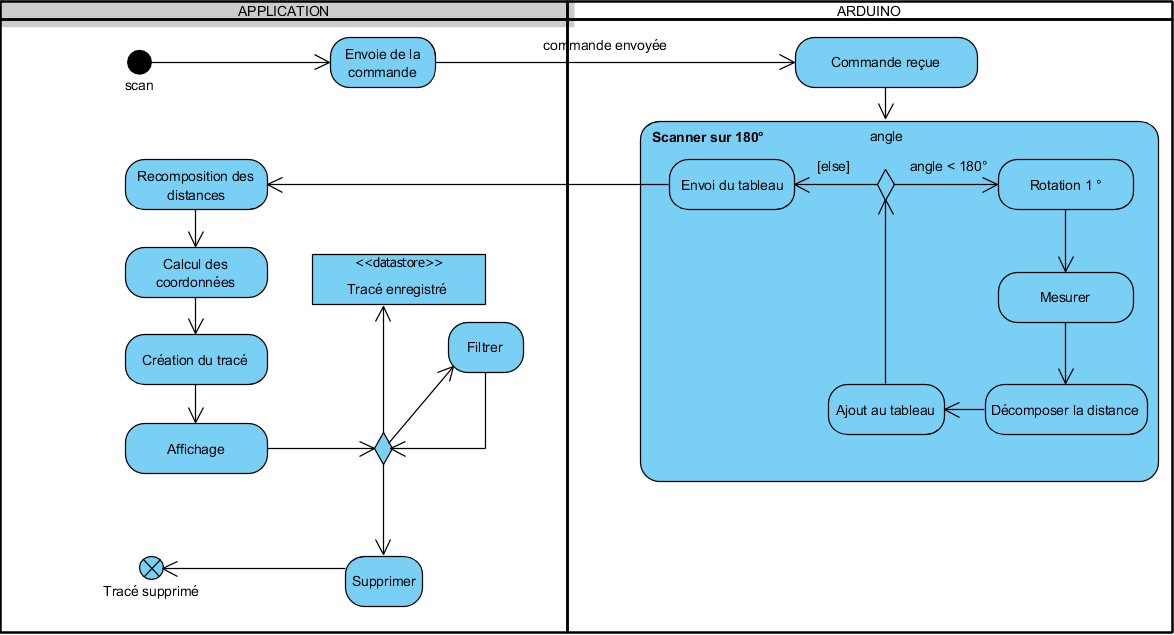
Le presenter reçoit ces deux valeurs et décide en réponse la direction dans laquelle le robot doit se déplacer :



1. Scanner

Lorsque l’utilisateur presse le bouton Scanner, l’application envoie la commande correspondante vers le robot.

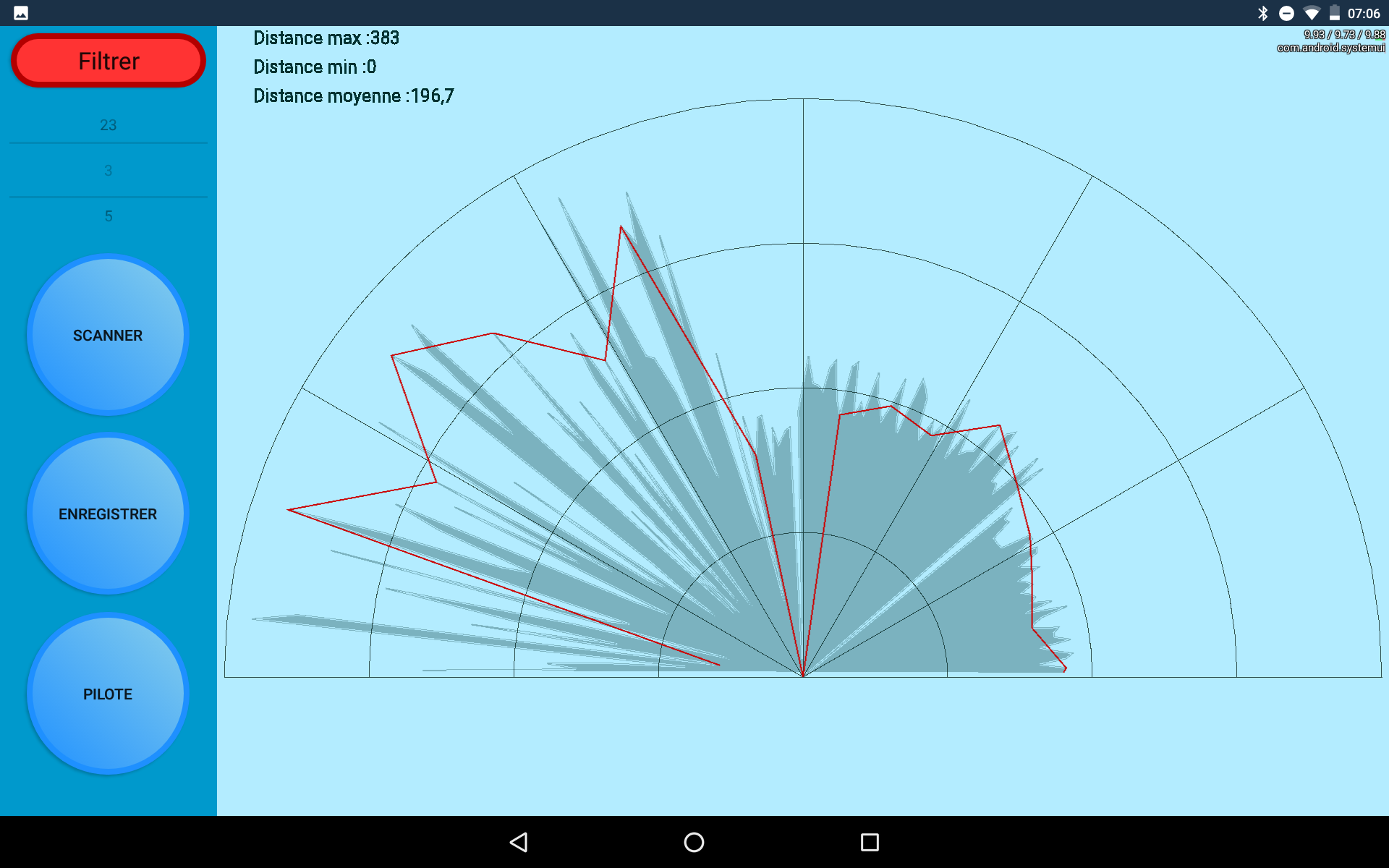
Cette action est volontairement bloquante pour éviter l’envoi d’autres commandes vers le robot pendant la durée du balayage, pendant celui-ci, le robot effectue une mesure de distance pour chaque degré d’angle sur un total de 180 degrés, un seconds balayage est réalisé et la moyenne des deux mesures est calculée, ces mesures sont stockées dans un byte array, qui est directement renvoyé vers le terminal.



Lorsque ce tableau de mesures est reçu dans l’application, un objet « ContoursData » est créé à partir de ces distances, cet objet contient aussi les coordonnées du nuage de points à afficher ainsi que quelques statistiques, distance minimum, maximum et moyenne.

Cet à partir de cette classe que le contour sera affiché sur l’écran.

Exemple de tracé, celui-ci a été généré par l’application elle-même pour des besoins de test, en gris apparaît le nuage de points calculés, en rouge, un tracé plus général ne passant que par un échantillon de points, aucun point n’est filtré.

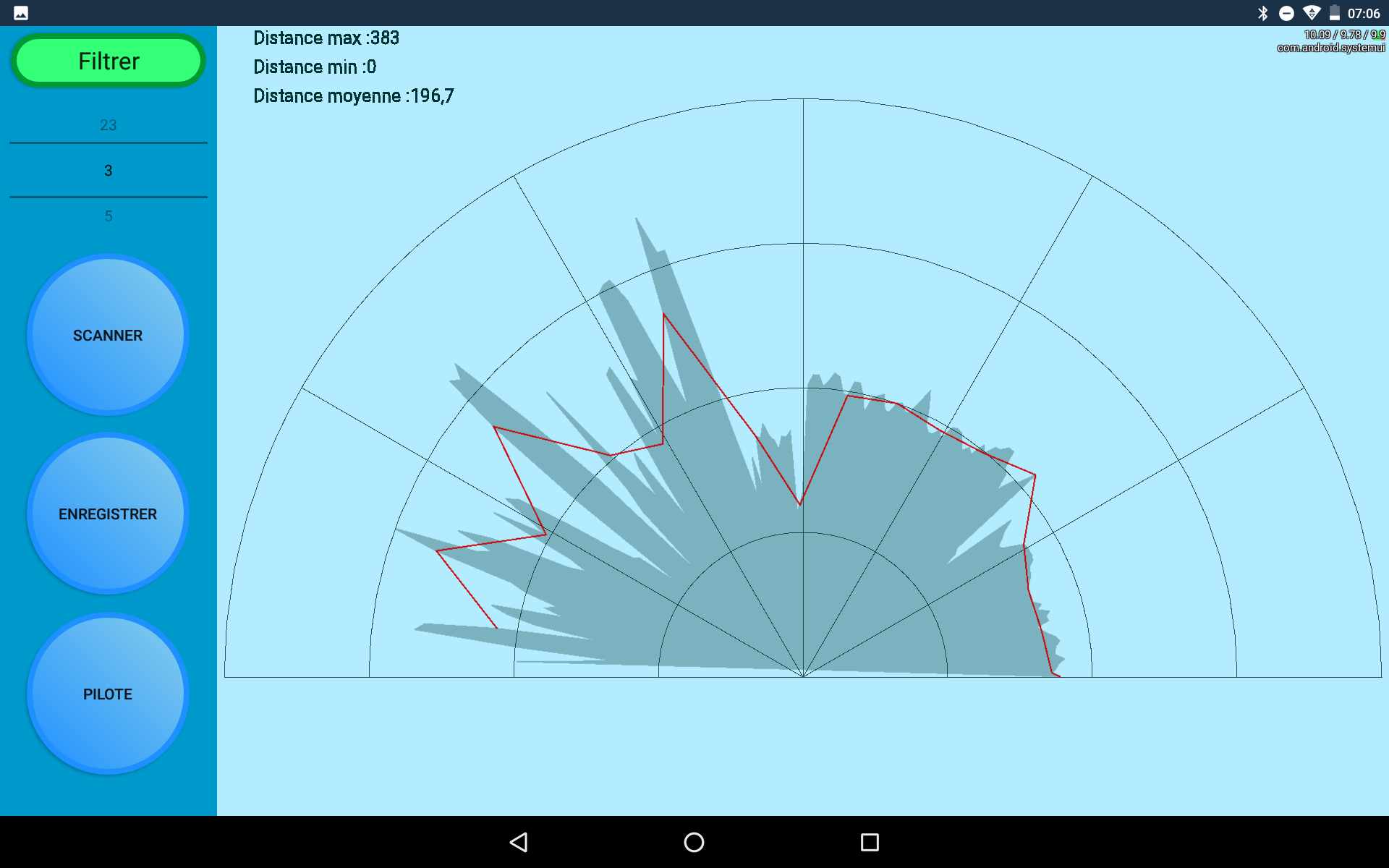


1. Filtrer

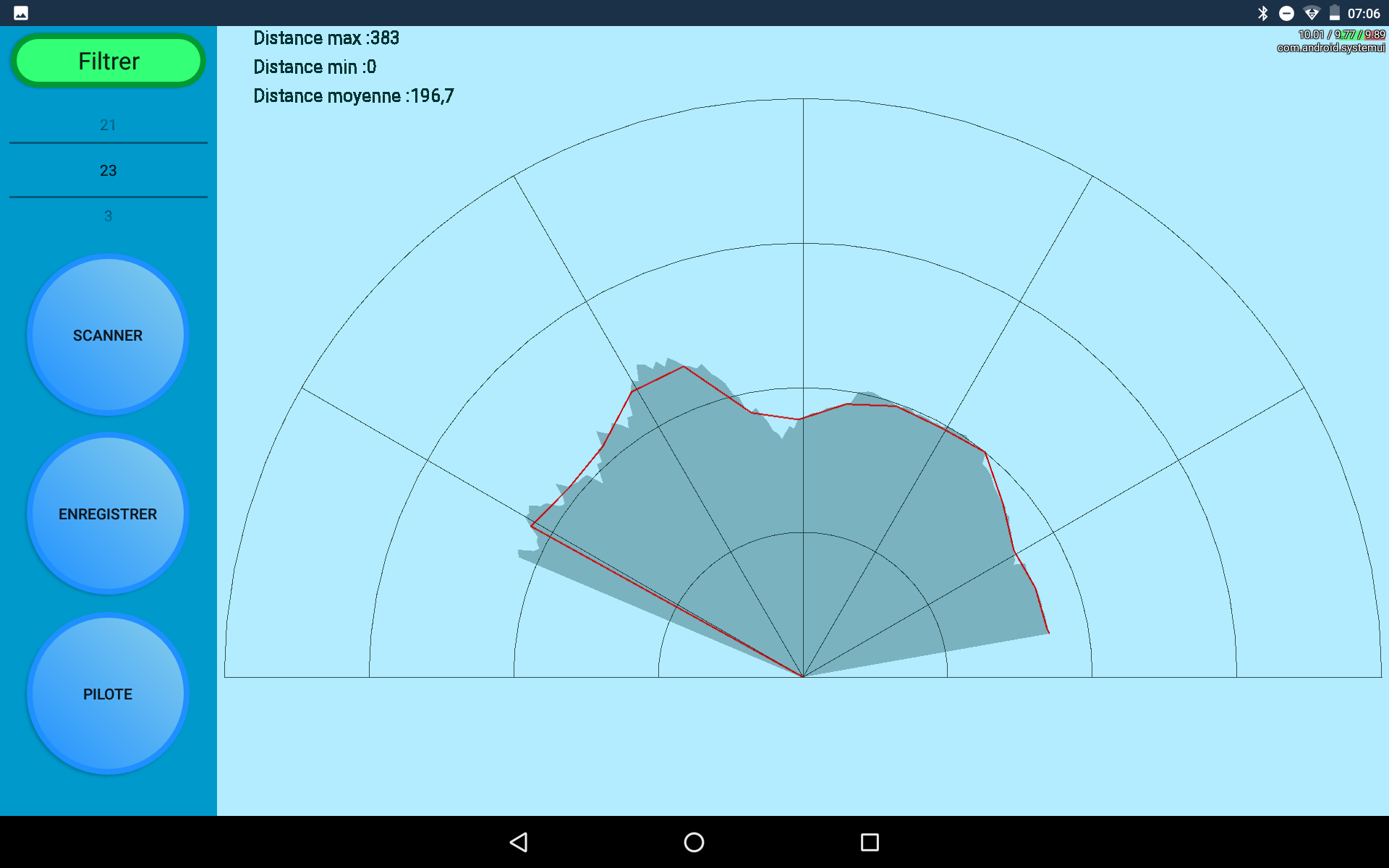
Lorsqu’un contours a été dessiné, il est possible d’utiliser une méthode de lissage sur ce tracé, celui-ci contient souvent des valeurs «aberrantes » pouvant être limitées pour ceci , une moyenne lissée peut être calculée sur un certain intervalle de points, allant de 3 à 23.

Le filtrage a été activé et une moyenne a été calculée sur un intervalle de trois points, le nuage de points et le tracé général apparaissent tout deux filtrés.

Les statistiques distance max, min et moyenne de changent pas, elles sont liées au calcul initial et pas au filtrage.



Un filtrage maximum, sur un intervalle de 23 points ne laisse qu’une représentation très générale du nuage de points.



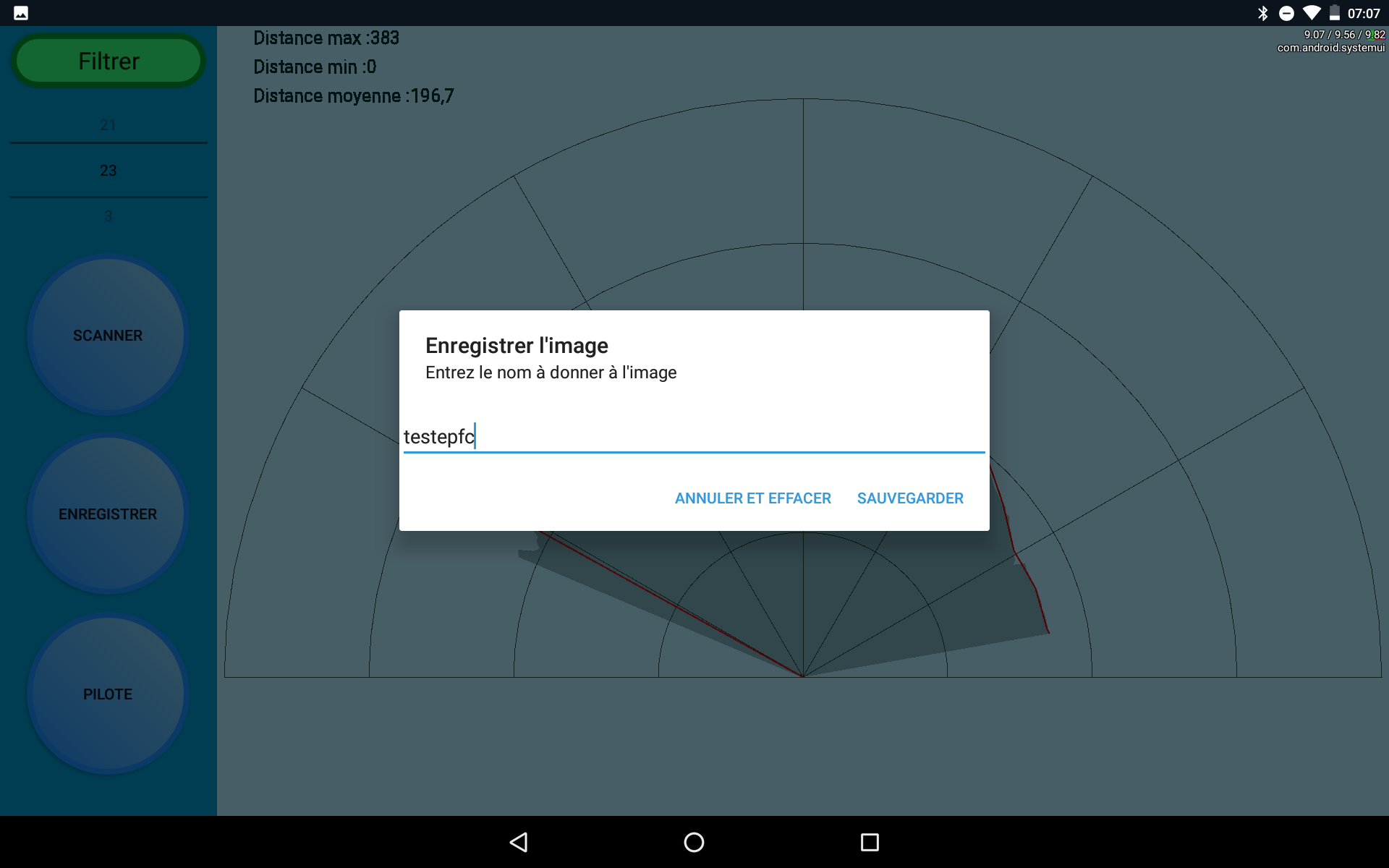
1. Enregistrer

Un clic sur le bouton d’enregistrement fait apparaître une boite de dialogue permettant soit d’enregistrer les données soit d’annuler et supprimer les données.

Il est toujours possible de revenir en arrière sans supprimer le tracé en appuyant su le bouton de retour d’Android.

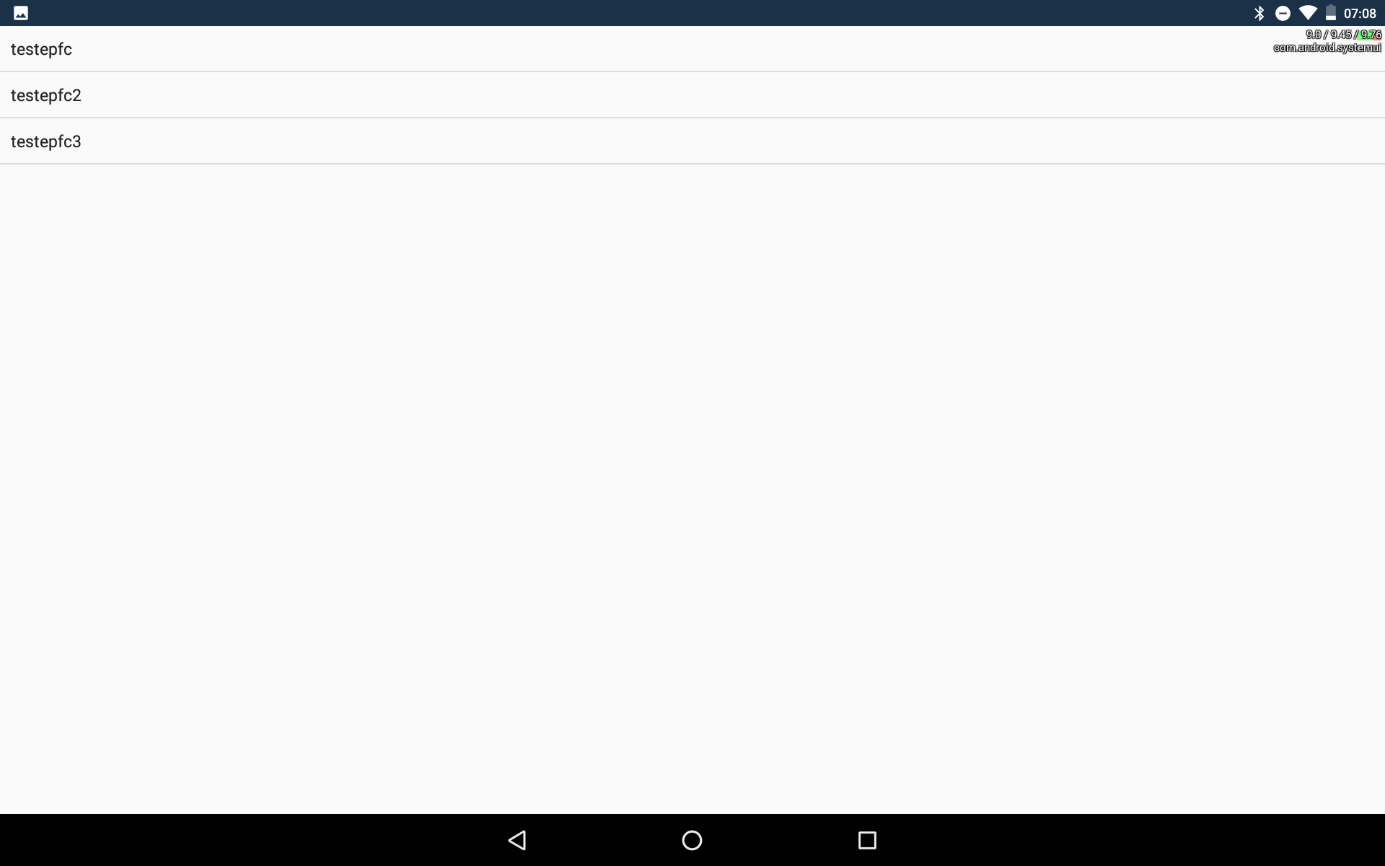
Lors d’un enregistrement, ce n’est pas l’image en elle-même qui est sauvegardée, mais bien le model servant à ce dessin, un objet « ContoursData » sérialisé au format Json et enregistrer dans le dossier de l’application, par défaut sous « Android.data.{nom de l’application} »

Ce format a l’avantage d’être facilement lisible et d’être devenu progressivement un format standard de partage d’information, offrant de cette manière une possibilité d’évolution de l’application, à savoir le partage des données vers d’autres terminaux/applications.



Ecran listing des contours

Le troisième écran de l’application est l’écran de listing des contours qui ont été enregistrés, cet écran permet au choix de supprimer un contour ou de l’afficher.



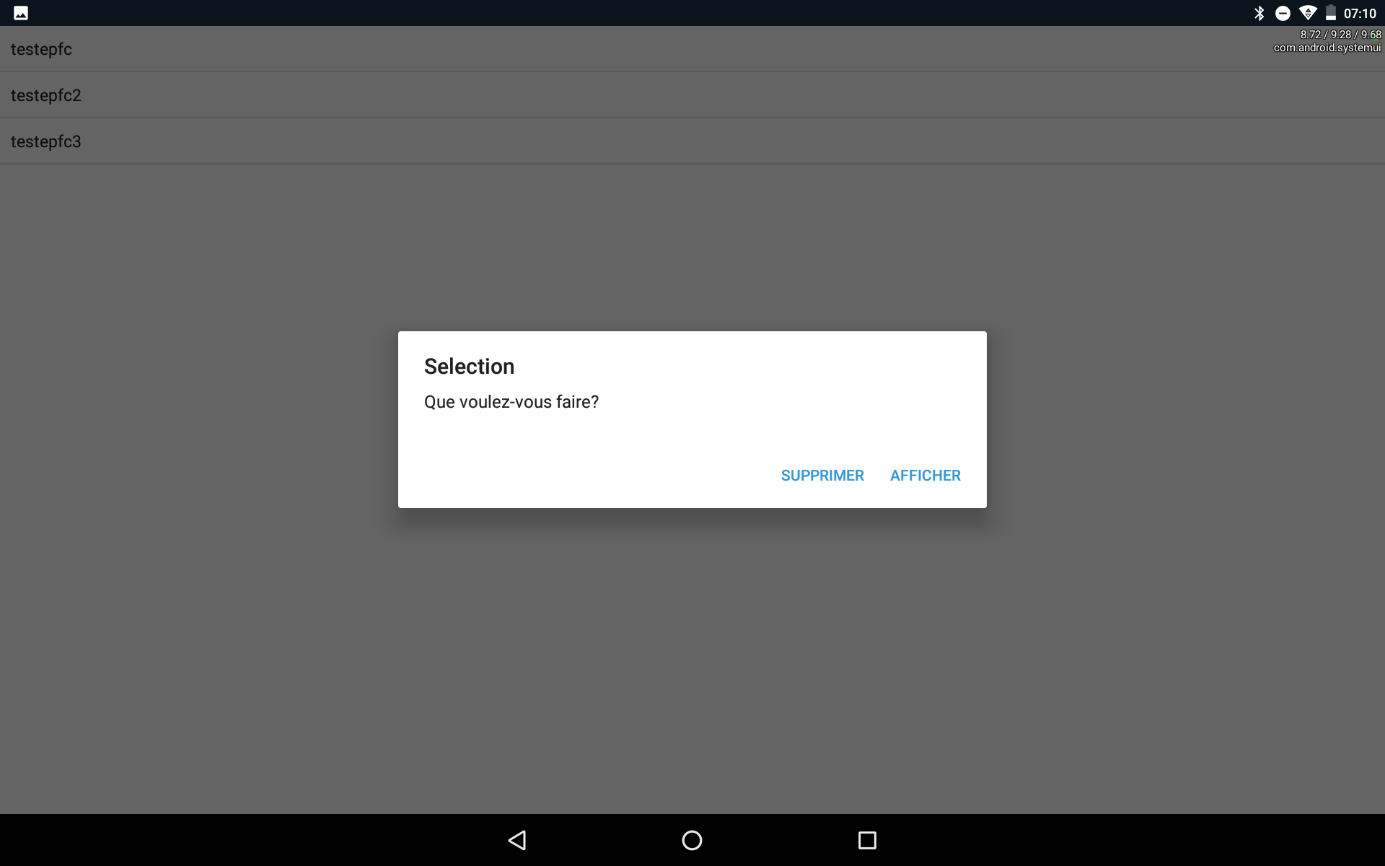
Fonctionnalités

1. Afficher/Supprimer

L’affichage du contour choisis se fait à nouveau dans l’écran de contrôle, cependant à ce moment, il n’y a aucune connexion avec le robot, les boutons scanner et piloter sont désactivé lors d’une revue d’un contour enregistré.

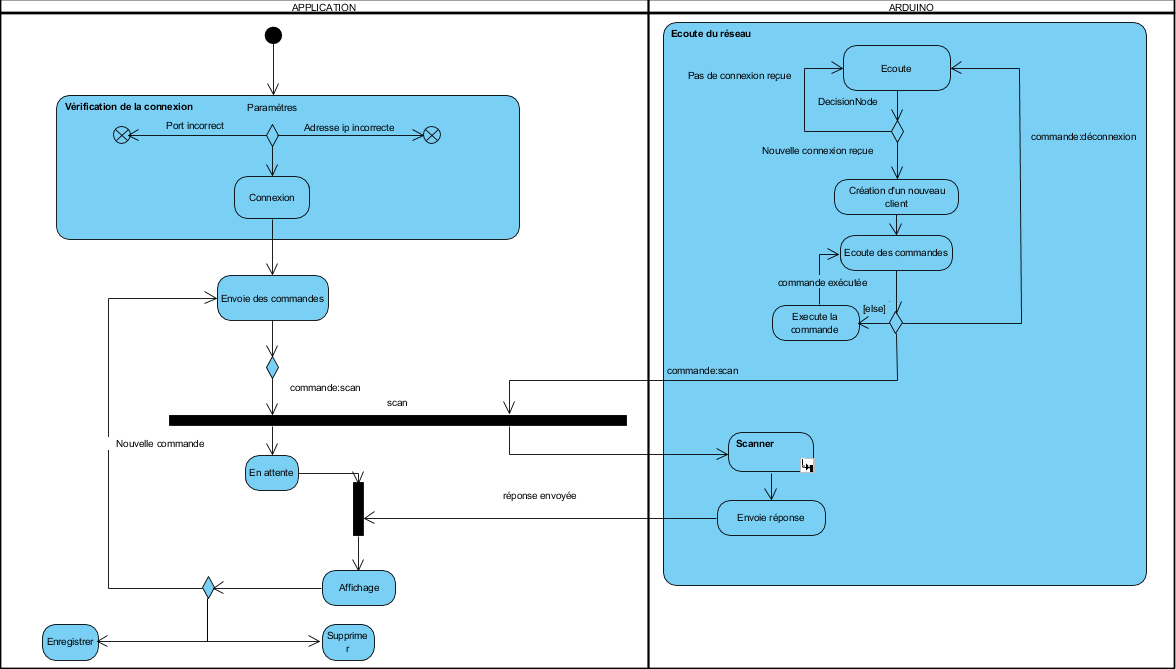
L’utilisateur a toujours la possibilité de filtrer, enregistrer et supprimer le tracé.

La suppression ne peut pas être annulé, le fichier contenant la représentation de l’objet est directement supprimé.



Communication wifi

La communication entre le terminal et le robot se fait en wifi, le microcontrôleur arduino possède un mode point d’accès wifi qui accepte les connexions du terminal, l’adresse ip ainsi que le port de connexion sont définit par défaut dans le programme arduino, il est toujours possible de les modifier depuis ce programme.



1. Scanner

1. https://fr.wikipedia.org/wiki/Microcontr%C3%B4leur [↑](#footnote-ref-1)
2. http://www-igm.univ-mlv.fr/~dr/XPOSE2002/robotique/chapitres/MicrocontroleurWhat.htm [↑](#footnote-ref-2)
3. https://makezine.com/2015/12/04/admittedly-simplistic-guide-raspberry-pi-vs-arduino/ [↑](#footnote-ref-3)
4. https://wiki.mchobby.be/index.php?title=HC-SR04 [↑](#footnote-ref-4)
5. https://fr.wikipedia.org/wiki/Microsoft\_Visual\_Studio#Visual\_Studio\_2017 [↑](#footnote-ref-5)
6. https://docs.microsoft.com/en-us/visualstudio/cross-platform/visual-studio-and-xamarin?view=vs-2015 [↑](#footnote-ref-6)
7. [↑](#footnote-ref-7)
8. Illustration https://android.jlelse.eu [↑](#footnote-ref-8)