**Phần I: Giới thiệu hệ điều hành Android**

**Hệ điều hành Android**

Android là một hệ điều hành mở dành cho thiết bị di động dựa trên một phiên bản được chỉnh sửa của Linux. Hệ điều hành này ban đầu được phát triển bởi một công ty cùng tên là Android Inc. Vào năm 2005, với chiến lược thâm nhập vào thị trường di động đầy tiềm năng, Google đã mua lại Android và tham gia vào việc phát triển và hoàn thiện nó.

Google muốn Android trở thành một hệ điều hành mở và hoàn toàn miễn phí, tuy nhiên, hầu hết mã nguồn của Android đều được cho ra mắt dưới bản quyền Apache dành cho mã nguồn mở, vì vậy bất cứ ai muốn sử dụng Android phải tải về toàn bộ mã nguồn của Android. Hơn thế nữa, các nhà sản xuất (đặc biệt là nhà sản xuất phần cứng) có thể thêm vào phần mở rộng của họ vào Android và tùy biến nó để trở nên khác biệt so với sản phẩm của hãng khác. Kiểu mẫu phát triển như vậy đã khiến cho Android trở nên rất hấp dẫn và thu hút nhiều nhà sản xuất. Điều này đặc biệt có ý nghĩa đối với những công ty đang bị ảnh hưởng bởi hiện tượng iPhone của Apple, một sản phẩm rất thành công tạo nên một cuộc cách mạng trong ngành công nghiệp Smartphone bấy giờ. Trong số những công ty đó, có Motorola và Sony Ericsson, những công ty cũng nhiều năm phát triển hệ điều hành di động của riêng mình. Khi iPhone ra mắt, những công ty đó phải tìm nhiều cách để có thể làm hồi sinh những sản phẩm của mình. Và họ đã tìm ra câu trả lời, chính là Android. Họ quyết định tiếp tục phát triển phần cứng của riêng mình, và sử dụng Android như là hệ điều hành.

Một trong những thuận lợi khi lựa chọn Android, đó là nó thay đổi cách tiếp cận của sự phát triển ứng dụng. Các nhà phát triển (Developer) chỉ phải phát triển ứng dụng cho Android, và các ứng dụng của họ sẽ có thể chạy trên vô số các thiết bị, miễn là sử dụng hệ điều hành Android. Trong thế giới Smartphone, các ứng dụng là phần quan trọng nhất trong dây chuyền thành công của sản phẩm. Các nhà sản xuất thiết bị coi Android là hi vọng lớn nhất của họ để có thể cạnh tranh với iPhone của Apple, một thiết bị đã có quá nhiều những ứng dụng lớn.

1. **Các phiên bản Android**

Kể từ khi phát hành chính thức, Google đã cho ra mắt rất nhiều phiên bản khác nhau của Android, có thể kể đến thời điểm hiện tại những phiên bản đã được ra mắt như dưới đây:

|  |  |
| --- | --- |
| Phiên bản | Tên mã |
| 1.1 |  |
| 1.5 | Cupcake |
| 1.6 | Donut |
| 2.0/ 2.1 | Éclair |
| 2.2 | Froyo |
| 2.3 | Gingerbread |
| 3.0/ 3.1/ 3.2 | Honeycomb |
| 4.0 | Ice Cream Sandwich |
| 4.1 | Jelly Bean |
| 4.2 | Jelly Bean |
| 4.3 | Jelly Bean |
| 4.4 | KitKat |

Kể từ phiên bản 1.5, các phiên bản Android lần lượt được đặt tên theo các loại bánh, theo thứ tự của bảng chữ cái. Phiên bản mới nhất vừa được ra mắt là phiên bản 4.4 với tên mã là KitKat.

Phiên bản KitKat ra mắt vào tháng 11/2013, từng bước hiện thực ước muốn của Google, đó là phổ biến hệ điều hành phiên bản mới nhất cho các dòng điện thoại có cấu hình ở mức trung bình. Google cho biết phiên bản KitKat sẽ có thể điều tiết lượng RAM tùy thuộc vào phần cứng của thiết bị, và KitKat có thể chạy trên những thiết bị có RAM là 512MB.

1. **Các tính năng của Android**

Vì Android là mã nguồn mở và hoàn toàn miễn phí đối với các nhà sản xuất để có thể tùy biến, nên không có một tùy chỉnh cấu hình phần cứng và phần mềm cố định nào. Tuy nhiên, bản thân Android hỗ trợ các tính năng sau:

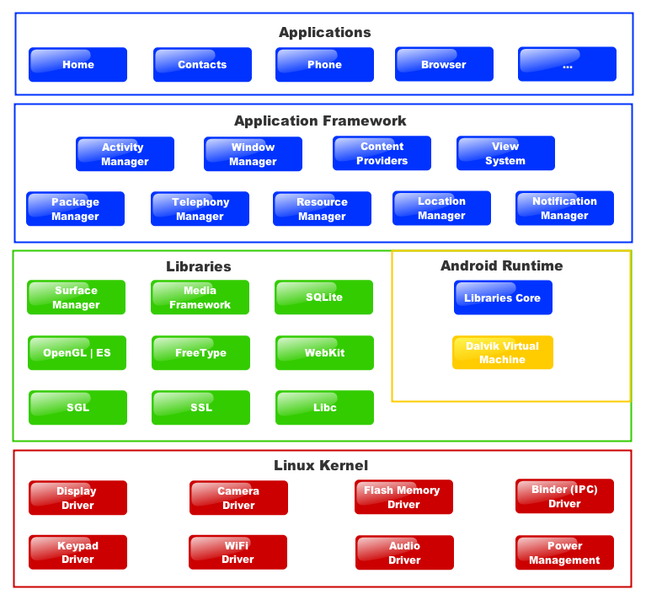
* Lưu trữ: Sử dụng SQLite để lưu trữ dữ liệu.
* Kết nối: Hỗ trợ GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (bao gồm A2DP và AVRCP), Wi-Fi, LTE và WiMAX.
* Tin nhắn: Hỗ trợ MMS và SMS.
* Trình duyệt: Dựa trên mã nguồn mở WebKit, cùng với Engine V8 JavaScript của Chrome.
* Hỗ trợ đa phương tiện: Hỗ trợ các chuẩn media: H.263, H.264 (ở định dạng 3GP hoặc MP4), MPEG-4 SP, AMR, AMR-WB (ở định dạng 3GP), AAC, HE-AAC (ở định dạng MP4 hoặc 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF và BMP.
* Hỗ trợ phần cứng: Cảm biến gia tốc (Accelerometer Sensor), Camera, La bàn số (Digital Compass), Cảm biến tiệm cận (Proximity Sensor) và GPS.
* Đa chạm: Hỗ trợ màn hình đa chạm.
* Đa nhiệm: Hỗ trợ những ứng dụng đa nhiệm.
* Hỗ trợ Flash: Android 2.3 hỗ trợ Flash 10.1.
* Phát sóng (Tethering): Hỗ trợ chia sẻ kết nối Internet như là một trạm phát không dây (Wireless hotspot).

1. **Cấu trúc của Android**

Để có thể hiểu cách thức vận hành của Android, ta sẽ cùng tìm hiểu các lớp (layers) mà tạo nên hệ điều hành Android.

Hệ điều hành Android được chia thành năm phần với bốn lớp chính:

* **Linux Kernel**: đây là kernel (nhân) mà Android dựa trên. Lớp này bao gồm toàn bộ những drivers thiết bị cấp thấp cho các thành phần phần cứng đa dạng của thiết bị Android.
* **Libraries**: Lớp này bao gồm toàn bộ code mà cung cấp những tính năng chính của Android. Ví dụ thư viện SQLite cung cấp hỗ trợ cơ sở dữ liệu mà nhờ đó ứng dụng có thể sử dụng để lưu trữ dữ liệu, hoặc thư viện WebKit cung cấp những tính năng để duyệt web.
* **Android runtime**: Ở cùng lớp với Libraries, Android runtime cung cấp một bộ thư viện lõi mà giúp các nhà phát triển có thể viết ứng dụng Android bằng ngôn ngữ lập trình Java. Android runtime còn bao gồm máy ảo Dalvik, giúp cho mỗi ứng dụng Android có thể chạy một process riêng bằng một Instance của máy ảo Dalvik (các ứng dụng Android được biên dịch thành các file thực thi Dalvik). Dalvik là một máy ảo được thiết kế đặc biệt cho Android và được tối ưu cho các thiết bị dùng pin với bộ nhớ và CPU hạn chế.
* **Android Framework**: Chỉ ra những khả năng của hệ điều hành Android, từ đó các nhà phát triển Android có thể sử dụng chúng trong những ứng dụng của mính.
* **Applications**: Ở lớp trên cùng này, ta có thể thấy những ứng dụng đi kèm với thiết bị Android (Danh bạ, Trình duyệt, vv) cũng như những ứng dụng ta download và cài đặt từ Market. Tất cả những ứng dụng mà ta viết đều nằm ở lớp này.



1. **The Android Market**

Giống như đã nói ở trên, yếu tố chính quyết định đến sự thành công của một nền tảng di động chính là những ứng dụng hỗ trợ nó. Từ sự thành công của iPhone có thể thấy rất rõ ràng rằng những ứng dụng có vai trò vô cùng quan trọng trong việc quyết định một nền tảng mới sẽ tồn tại hay bị diệt vong. Thêm vào đó, việc làm cho những ứng dụng này đến với những người dùng phổ thông là hết sức quan trọng.

Vì vậy, vào tháng 8/2008, Google thông báo sẽ mở ra Android Market, một kho ứng dụng online cho những thiết bị Android, và mở cửa vào tháng 10/2008. Sử dụng ứng dụng Market được cài đặt sẵn, người dùng có thể download những ứng dụng của bên thứ ba một cách đơn giản. Market hỗ trợ cả ứng dụng miễn phí và trả phí, mặc dù một vài ứng dụng trả phí chỉ dành cho những người dùng ở những quốc gia nhất định vì lí do pháp lí.

1. **Những công cụ cần thiết**

Để có thể phát triển ứng dụng Android, ta cần download những công cụ cần thiết và các bộ SDK.

Để lập trình Android, ta có thể sử dụng một máy tính chạy hệ điều hành Windows, MAC hoặc Linux. Tất cả những công cụ đều miễn phí và có thể download trực tiếp từ Web.

Trước khi download Android SDK, ta cần cài đặt Java JDK, vì Android SDK sử dụng Java SE Development Kit (JDK). Ta có thể download dễ dàng trên trang chủ của Oracle: www.oracle.com/technetwork/java/javase/downloads/index.html

Tiếp theo sau khi đã download và cài đặt JDK, ta sẽ download và cài đặt phần mềm quan trọng nhất, chính là Android SDK. Android SDK bao gồm debugger, các thư viện, bộ mô phỏng emulator, các tài liệu, code mẫu và các bài hướng dẫn.

Ta có thể download Android SDK trên website dành cho nhà phát triển của Android: http://developer.android.com/sdk/index.html

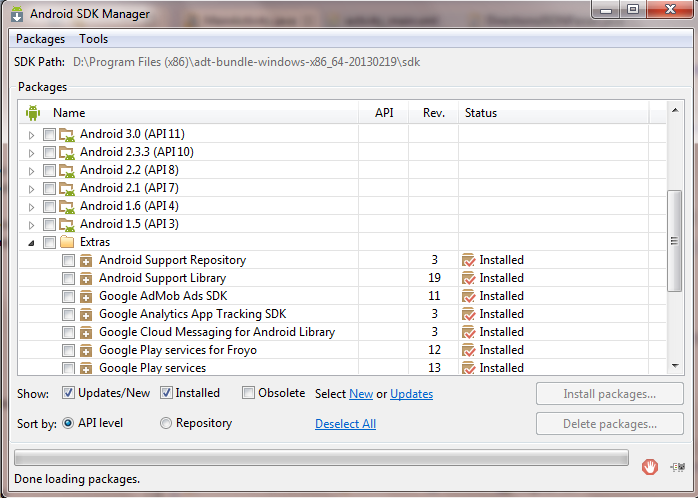
Sau khi download xong Android SDK, ta tiến hành cài đặt, lưu ý khi chọn địa chỉ lưu bộ SDK.

1. **Tùy chình Android SDK Manager**

Android SDK Manager là một phần mềm quản lý các phiên bản Android SDK hiện tại được cài đặt trong máy. Khi khởi động nó sẽ cho thấy một danh sách các mục và các mục đó đã được cài đặt hay chưa.

Đánh dấu vào các mục ta cần download, thường thì bộ SDK download từ ở trên sẽ chỉ gồm Platform phiên bản Android mới nhất, nếu muốn lập trình trên những phiên bản Android thấp hơn, ta phải chọn download các Platform còn thiếu từ SDK Manager.

Đề tài luận văn là “Lập trình xe tự hành dựa trên Google Maps API”, nên phần quan trọng không thể thiếu là bộ SDK phải hỗ trợ Google Maps API. API này nằm trong Google Play Service, nên ta phải download mục này thì mới có thể lập trình được.

Hình ảnh của Android SDK Manager:

1. **Eclipse**

Bước tiếp theo là cài đặt môi trường phát triển tích hợp (IDE) để phát triển ứng dụng Android. Và Google khuyến khích sử dụng Eclipse, một IDE hỗ trợ lập trình nhiều ngôn ngữ phổ biến như Java, Ada, C, C++, COBOL, Python, …

Để lập trình Android, ta nên download Eclipse IDE for Java EE Developers. Ta phải chọn hệ điều hành phù hợp để download phiên bản Eclipse phù hợp cho mình. Trong trường hợp này là Windows 64 bit.

Các bước download và cài đặt khá đơn giản. Chỉ cần giải nén file download được là có thể sử dụng được Eclipse bằng cách mở file thực thi eclipse.exe.

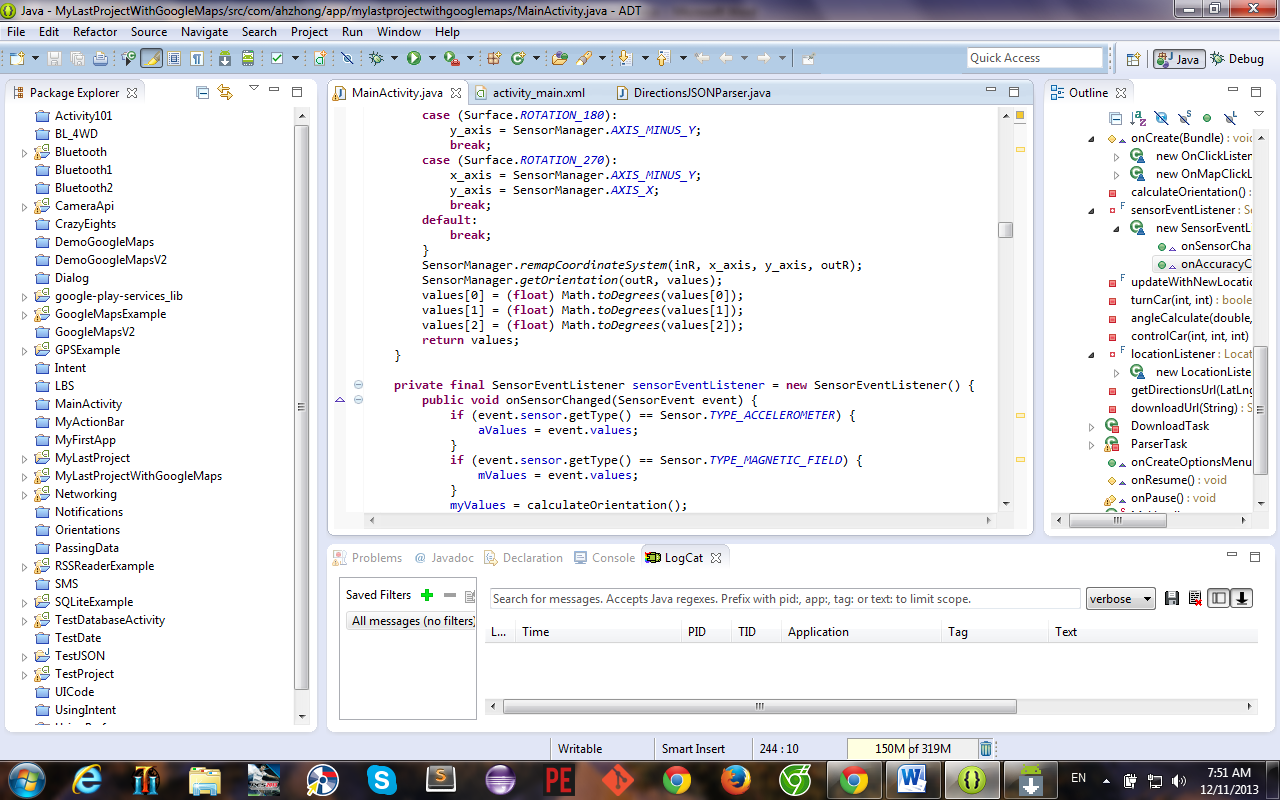
Đối với những bản Android SDK sau này (từ 4.2 trở đi), thì Google đã giúp tích hợp Eclipse vào bộ Android SDK, khi chúng ta giải nén Android SDK thì cũng có sẵn một thư mục Eclipse, ta không cần phải download thêm ở ngoài nữa.

Thêm vào đó, việc tích hợp sẵn Eclipse vào bộ cài đặt Android SDK cũng giảm nhiều công sức cho người dùng, vì phiên bản Eclipse này đã được cài đặt sẵn phần mở rộng ADT, mà ngày xưa người dùng phải tự cài đặt thêm khi cài đặt Eclipse.

Tiện ích mở rộng ADT sẽ giúp chúng ta:

* Tạo project Android mới.
* Tiếp cận với các công cụ để tiếp cận bộ Emulator hoặc thiết bị Android của chúng ta.
* Biên dịch và debug chương trình Android.
* Export ứng dụng Android (file APK).
* Tạo chứng chỉ số cho chữ kí mã hóa cho file APK.

Hình ảnh của Eclipse:



1. **Tạo thiết bị ảo Android (Android Virtual Devices – AVD)**

Một phần không thể thiếu của bộ công cụ lập trình ứng dụng Android, chính là thiết bị ảo Android – AVD. AVD được Google tích hợp vào bộ Android SDK nhằm giúp người dùng có thể chạy thử các ứng dụng của họ. AVD có thể mô phỏng hoạt động của một thiết bị thực. Mỗi AVD có một cấu hình phần cứng, file hệ thống cũng như bộ nhớ (vd như SD card).

Ta có thể tạo nhiều AVD để có thể chạy thử chương trình trên nhiều cấu hình phần cứng và phần mềm khác nhau. Việc chạy thử này rất quan trọng để kiểm tra ứng dụng có hoạt động đúng trên nhiều thiết bị có cấu hình khác nhau hay không.

Trong Eclipse, ta tạo AVD bằng cách vào Window -> AVD Manager, chọn New ở cửa sổ Android Virtual Device Manager, cửa sổ mới hiện lên sẽ cho ta tùy chình AVD.

Khi lập trình ứng dụng thì ta nên tạo nhiều Android với nhiều cấu hình phần cứng và phần mềm khác nhau để kiểm tra sự tương thích và hoạt động của ứng dụng.

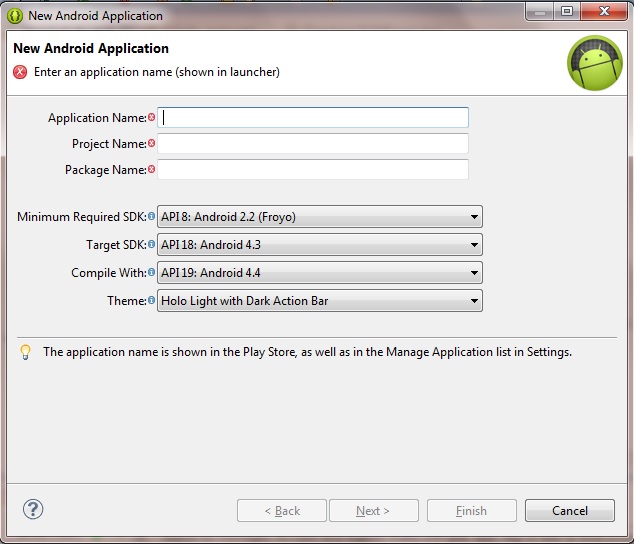
Tuy nhiên, AVD cũng có một vài nhược điểm:

* Vì là một thiết bị ảo Android, nên dẫn đến AVD chạy không được mượt như thiết bị thực, thậm chí đáp ứng rất chậm, gây khó khăn trong việc chạy thử ứng dụng.
* AVD không thể mô phỏng những ứng dụng đòi hỏi sử dụng phần cứng của điện thoại, ví dụ như Bluetooth, Sensor, …

Trong phạm vi của luận văn, rõ ràng không thể sử dụng AVD để mô phỏng, do luận văn sử dụng nhiều tính năng phần cứng của một thiết bị thực như Bluetooth và các Sensor, vì vậy chỉ có thể chạy trên một thiết bị thực mới có thể kiểm tra được ứng dụng.

1. **Tạo một ứng dụng Android trong Eclipse**

Trong Eclipse, mở File -> New -> Android Application Project.



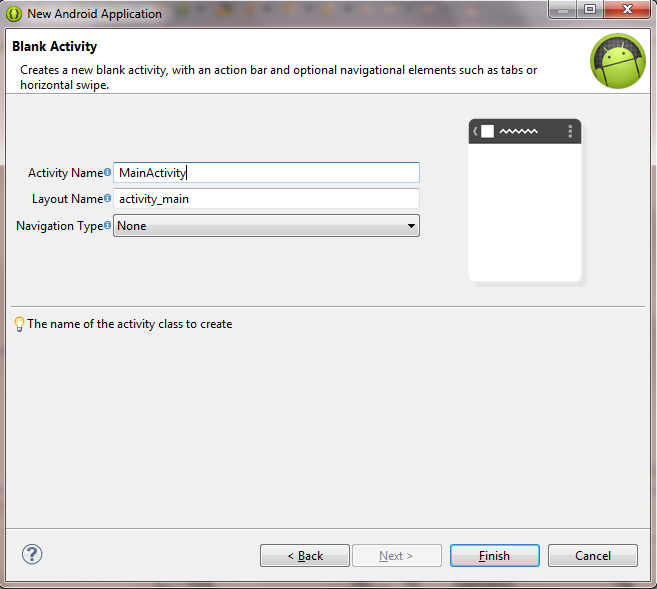
Sau đó nhập các field yêu cầu:

* Application Name: tên của ứng dụng, là tên mà người dùng nhìn thấy, nên có thể có khoảng trắng và mô tả ứng dụng.
* Project Name: tên của project trong Eclipse, và bắt buộc đặt tên theo quy định của một project trong Eclipse (không được có khoảng trắng và các kí tự đặc biệt).
* Package Name: tên của package chứa các Class của project. Trong phạm vi luận văn này, package name còn dùng để lấy API key mà Google cung cấp cho các ứng dụng sử dụng Google Maps API.
* Minimum Required SDK: nền tảng Android thấp nhất mà ứng dụng hỗ trợ.
* Target SDK: là nền tảng Android mà ứng dụng hướng đến, có thể coi là nền tảng Android cao nhất mà ứng dụng hỗ trợ.
* Compile with: là nền tảng Android dùng để biên dịch ứng dụng, thường chọn bằng với Target SDK.

Sau khi thiết lập xong ta chọn Next, ở hai bước tiếp theo là các công đoạn lựa chọn các tùy chọn (như workspace, icon), ta có thể để mặc định hai bước này và bấm Next để đến bước tiếp theo.

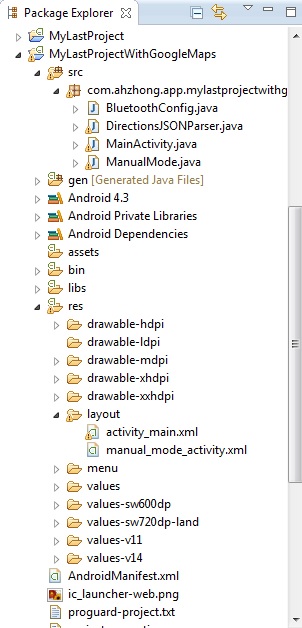
Bước tiếp theo là Create Activity, Activity có thể hiểu là một Class để định nghĩa hành vi của ứng dụng khi tương tác với người dùng, có thể coi đơn giản Activity là một cửa sổ để người dùng có thể tương tác thông qua các đối tượng. Ở mục tiếp theo bên dưới sẽ nói thêm về Activity trong một ứng dụng Android.

Cửa sổ này ta có thể để mặc định dấu check ở Blank Activity, nghĩa là Eclipse sẽ tạo cho ta một Activity trống. Ta bấm Next qua bước tiếp theo.



Ở cửa sổ tiếp theo, ta sẽ điền vào hai field:

* Activity Name: tên của Activity, thông thường trong một chương trình Android sẽ có rất nhiều Activity khác nhau, tương ứng với những cửa sổ mà người dùng tương tác, Activity Name nên đặt theo behavior của nó (ví dụ như Activity của mode manual nên đặt tên là ManualModeActivity).
* Layout Name: vì Activity định nghĩa các Class dùng để tương tác với người dùng, nên ứng với mỗi Activity sẽ có một Layout dùng để mô tả giao diện cho nó, ta cũng nên đặt tên Layout có liên quan đến Activity mà nó mô tả.



1. **Cấu trúc của một ứng dụng Android**

Chúng ta sẽ cùng xem xét những file và folder được tạo ra trong quá trình phát triển ứng dụng Android:

* Src: là thư mục chứa các file mã nguồn .java cho project, ta sẽ thấy các file này nằm trong package mà ta đặt tên lúc đầu, ngoài các Activity ra ta có thể tạo thêm nhiều Class thông thường khác để định nghĩa các Config cần thiết cho project.
* Gen: Chứa file R.java, một file tạo bởi bộ biên dịch để tham chiếu đến tất cả tài nguyên trong project. Ta không nên chỉnh sửa file này, vì tất cả các tài nguyên sẽ được tự động biên dịch vào file này và ta có thể tham chiếu đến nó.
* Android 4.3: chứa file android.jar, chứa toàn bộ các thư viện cần thiết cho ứng dụng Android.
* Assets: chứa những asset dùng bởi ứng dụng, như HTML, database hoặc file text, …
* Bin: folder này chứa các file được build bởi ADT (Android Development Tool), và đây cũng là nơi tạo ra file .apk (Android Package). File .apk là file thực thi của Android, nó chứa mọi thứ cần để chạy được ứng dụng Android. Có thể coi file .apk tương đương với file .exe của hệ điều hành Windows.
* Res: folder này chứa tất cả tài nguyên (resources) của project. Folder này có một vài sub-folder như drawable - <resolution> dùng để chứa các file ảnh, layout dùng để chứa các file layout, values dùng để chứa các giá trị (như string, array, …)
* AndroidManifest.xml: đây là file khai báo của project. Ta sẽ phải khai báo toàn bộ các permission (ví dụ như Internet permission, Bluetooth permission, …) và khai báo toàn bộ các Activity dùng trong chương trình.

Hình ảnh ví dụ của AndroidManifest.xml



Trong một file AndroidManifest.xml, trước hết vì đây là định dạng file XML, nên nó sẽ có cấu trúc bao gồm các thẻ <element></element>, và bên trong có những element khác lồng vào để định nghĩa các element lớn hơn.

* AndroidManifest.xml trước hết khai báo Package Name của project đang thực hiện trong thuộc tính package của thẻ <manifest>.
* Tiếp theo là khai báo mã phiên bản (version code), khai báo này nhằm mục đích để xác định xem một ứng dụng đã được cài sẵn có cần nâng cấp hay không.
* Tiếp đến là tên phiên bản (version name), đây là tên được hiển thị tới người dùng, thường đặt theo cú pháp là <major>.<minor>.<point>.
* Tiếp đến là thuộc tính minSdkVersion của thẻ <uses-sdk>, dùng để khai báo phiên bản Android thấp nhất mà ứng dụng hỗ trợ.
* Ứng dụng sẽ lấy ảnh ic\_launcher.png nằm ở thư mục drawables để làm icon.
* Trong ví dụ này, chương trình chỉ có một Activity nên ta chỉ khai báo một Activity này trong file AndroidManifest.xml

Khi ta khai báo trong file Manifest và tạo các folder, các file, thì lập tức Eclipse sẽ tự động build file R.java. Ta cùng xem ví dụ một file R.java như bên dưới:



Như file ví dụ ở trên, khi ta thêm các file hoặc folder vào project, Eclipse sẽ tự động chỉnh sửa file R.java, và chúng ta không nên chỉnh sửa file này.

**Giới thiệu Activity**

Như đã giới thiệu ở trên, một Activity là một cửa sổ chứa các giao diện người dùng (User Interface) của ứng dụng Android. Một ứng dụng có thể không có hoặc có nhiều Activity khác nhau. Thông thường, các ứng dụng có một hoặc nhiều Activity, và mục đích chính của Activity là để tương tác với người dùng.

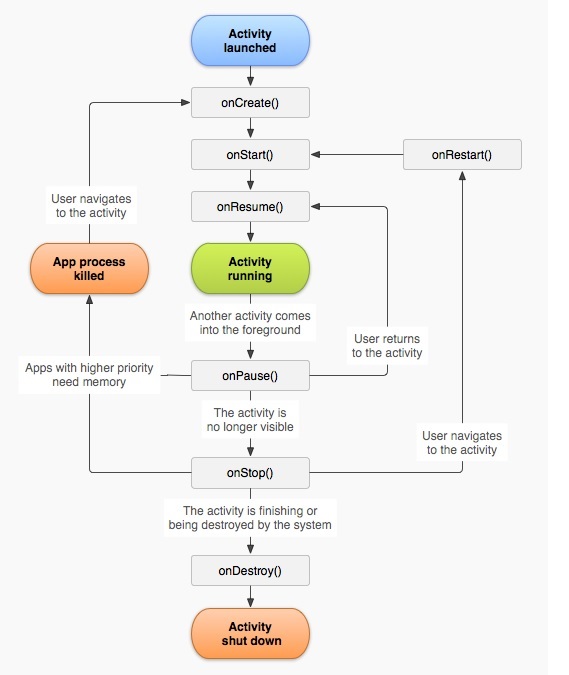
1. **Life cycle của một Activity**

Từ lúc Activity xuất hiện trên màn hình cho đến khi bị che khuất đi, nó trải qua nhiều giai đoạn, thường được gọi là Life Cycle của Activity. Hiểu rõ Life Cycle của Activity là một điều quan trọng để đảm bảo rằng ứng dụng của chúng ta hoạt động đúng. Các Activity trong hệ thống được quản lý theo các “Activity Stack”. Khi một Activity mới được bắt đầu, nó sẽ được đặt trên cùng của Stack, và trở thành Activity được thực thi, khi đó Activity trước đó sẽ nằm bên dưới trong Stack, và chỉ xuất hiện trở lại màn hình chính khi Activity mới thoát ra.

Một Activity có bốn trạng thái chính:

* Nếu một Activity ở trên màn hình chính (ở trên cùng của Stack), ta nói Activity đó đang “Active” hoặc “Running”.
* Nếu một Activity không còn ở màn hình chính nữa nhưng vẫn nhìn thấy được (một Activity không full-screen mới đã xuất hiện ở trên cùng của Stack), ta nói nó đã bị “Paused”. Một Activity bị Paused thực chất vẫn đang hiện hữu (nó giữ nguyên các trạng thái, thông tin của các thành viên và vẫn gắn với Window Manager), nhưng khi trạng thái thiếu bộ nhớ thì nó sẽ bị triệt tiêu.
* Khi một Activity bị che lấp hoàn toàn bởi một Activity mới, ta nói nó bị “Stopped”. Nó vẫn giữ nguyên các trạng thái và thông tin của thành viên nhưng cửa sổ của nó đã bị che lấp hoàn toàn và nó dễ bị triệt tiêu bởi hệ thống khi hệ thống cần bộ nhớ.
* Nếu một Activity bị “Paused” hoặc “Stopped”, nó có thể bị hệ thống triệt tiêu để giải phóng bộ nhớ (bằng cách bắt buộc Activity hoàn thành, hoặc triệt tiêu các process của nó). Khi Activity này được hiển thị trở lại, nó phải được khởi động lại hoàn toàn và khôi phục lại các trạng thái trước đây.

Biểu đồ sau diễn tả các trạng thái quan trọng của Activity. Các hình chữ nhật biểu thị các hàm (Callback Method) mà ta có thể lập trình khi Activity chuyển trạng thái. Những hình oval có màu là các trạng thái chính của các Activity.



Toàn bộ Life Cycle của một Activity được định nghĩa bởi các hàm bên dưới. Ta hoàn toàn có thể Override lại các hàm này để ứng dụng có thể làm việc như mong muốn trong quá trình chuyển trạng thái. Tất cả Activity sẽ phải chạy hàm onCreate(Bundle) để setup ban đầu, và thông thường cả hàm onPause() để xác thực các thay đổi đến dữ liệu hoặc chuẩn bị dừng tương tác với người dùng.

Và thông thường sự chuyển dịch giữa các Activity có thể được mô tả như bên dưới:





1. **Intent và bắt đầu một Activity mới**

Khi muốn bắt đầu một Activity mới, ta sẽ gọi hàm  startActivity(Intent). Activity được gọi sẽ nằm ở trên cùng của Stack. Hàm này cần một tham số, đó là một đối tượng “Intent”, dùng để mô tả Activity mới.

Một Intent thực chất có thể coi như là một thứ kết dính các Activity lại với nhau, dù có thể các Activity này không cùng nằm trong một ứng dụng. Thông thường ta dùng để gọi đến các Activity trong cùng một ứng dụng, hoặc các Activity mặc định của hệ thống (ví dụ như Trình duyệt, Bản đồ, Bluetooth, …).

Đôi khi ta muốn nhận một kết quả trả về từ Activity mà ta gọi (khi nó kết thúc và trả Activity cũ về vị trí trên cùng của Stack). Ví dụ như ta gọi một Activity của hệ thống để xin phép bật Bluetooth, và sau đó ta muốn biết là người dùng có cho phép hay không, bằng cách đọc kết quả mà Activity đó trả về. Để làm được điều đó, thay vì gọi hàm startActivity(Intent) thì ta sẽ gọi hàm   startActivityForResult(Intent, int), với tham số thứ hai để nhận diện sự gọi đến Activity đó. Kết quả trả về sẽ gọi hàm  onActivityResult(int, int, Intent).

1. **Giao diện người dùng**

Trong các chương trình Android, một Activity thường có mục đích chính là để tương tác với người dùng. Vì vậy Android cung cấp cho chúng ta một Class Activity để ta có thể thừa kế và tùy biến cho các Activity của riêng mình.

Một trong những điều Class Activity mang lại cho chúng ta, là nó sẽ cung cấp hàm để tạo nên một cửa sổ cho người dùng, từ cửa sổ đó chúng ta có thể tùy biến Giao Diện Người Dùng (User Interface – UI). Hàm mà Class Activity cung cấp cho chúng ta là hàm “setContentView(View)”. Trong đó View là một Class được định nghĩa sẵn của hệ điều hành Android, mô tả thuộc tính cho các đối tượng View mà người dùng sẽ tương tác (có thể kể ra như Button, TextView, EditText, …).

Các Activity thường được hiển thị lên màn hình theo kích thước full-screen để có thể dễ dàng tương tác với người dùng, ta hoàn toàn có thể thay đổi các thể hiện mặc định này.

Và để định nghĩa giao diện người dùng (UI), ta sẽ sử dụng file XML (Layout Name) nằm trong thư mục res/layout.

Ví dụ của một file XML định nghĩa layout của một Activity:



Ở file này ta định nghĩa layout của Activity là kiểu LinearLayout, với các thuộc tính độ dài và độ rộng là “fill\_parent”, nghĩa là layout sẽ tràn ra toàn bộ phần nhìn thấy của Activity. Layout có một đối tượng TextView, với độ rộng là “fill\_parent”, độ cao là “wrap\_content”, nghĩa là sẽ phụ thuộc vào content của TextView đó. TextView này lấy giá trị ban đầu ở trong thư mục res/values/string.xml. Đây là nơi định nghĩa các giá trị String.

Sau khi định nghĩa file XML, ta sẽ set layout này vào Activity ngay khi Activity được khởi động, tức là ở hàm onCreate(Bundle):



Như đã giới thiệu, ta có R chính là đối tượng R định nghĩa trong R.java. Khi ta thay đổi những tài nguyên của project, lập tức file R.java sẽ được cập nhật. Khi ta thay đổi file layout, thì R sẽ cập nhật sự thay đổi đó. Nên ta dễ dàng set layout cho Activity bằng hàm setContentView(R.layout.main), với main là tên của file XML định nghĩa layout cho Activity này.

Trong Android, ta có những kiểu Layout sau:

* LinearLayout
* RelativeLayout
* AbsoluteLayout
* TableLayout
* FrameLayout
* ScrollView

Thường thì trong chương trình ta sẽ kết hợp nhiều kiểu layout cho từng nhóm đối tượng tương tác để tạo ra layout mong muốn.

**PHẦN II: PROJECT ANDROID THỰC HIỆN CHO LUẬN VĂN**

1. **Bluetooth**
2. **Giới thiệu**

Bluetooth là phương thức dùng để truyền tín hiệu điều khiển từ điện thoại đến vi điều khiển STM32F4, vì vậy trong chương trình Android ta cần phải config các tính năng của Bluetooth. Do Android lập trình trên ngôn ngữ Java, một ngôn ngữ lập trình hướng đối tượng, để có thể tận dụng được tối đa những lợi thế của một ngôn ngữ lập trình hướng đối tượng, ta sẽ tạo một Class BluetoothConfig.java, với mục đích chính là mô tả một đối tượng BluetoothConfig chung, và ở các Activity ta chỉ cần gọi Instance của nó mà không cần phải lặp đi lặp lại các bước Config.

Nền tảng Android hỗ trợ chuẩn truyền không dây Bluetooth. Các tính năng của Bluetooth được sử dụng trong các ứng dụng có thể được lập trình thông qua các Bluetooth APIs. Những API này cho phép các thiết bị kết nối Bluetooth và các tính năng truyền nhận không dây.

Sử dụng Bluetooth API, ứng dụng Android có thể:

* Tìm kiếm các thiết bị Bluetooth.
* Truy vấn các thiết bị Bluetooth đã bắt cặp (paired devices).
* Thành lập các kênh RFCOMM.
* Kết nối tới các thiết bị khác thông qua service discovery.
* Truyền nhận dữ liệu tới và từ các thiết bị khác.

Để có thể sử dụng những tính năng chính của Bluetooth như: setting up, tìm kiếm thiết bị, kết nối thiết bị và truyền nhận dữ liệu giữa các thiết bị, ta sẽ sử dụng các Class được định nghĩa sẵn trong package android.bluetooth. Trong phạm vi luận văn ta sẽ sử dụng các Class sau:

* BluetoothAdapter: Class này đại diện cho Bluetooth Adapter của thiết bị điện thoại chạy hệ điều hành Android. Sử dụng Class này ta có thể tìm kiếm những thiết bị khác, truy vấn những thiết bị đã bắt cặp sẵn, hoặc tạo một Instance BluetoothDevice với tham số là địa chỉ MAC của thiết bị đó, và tạo một BluetoothServerSocket để nhận dữ liệu từ thiết bị khác.
* BluetoothDevice: đại diện cho các thiết bị trong phạm vi. Sử dụng Class này để yêu cầu kết nối với thiết bị đó thông qua BluetoothSocket hoặc truy vấn thông tin của thiết bị như: tên, địa chỉ, trạng thái, …
* BluetoothSocket: đại diện một giao diện (Interface) cho một Bluetooth Socket (tương tự như một TCP Socket). Đây là điểm kết nối cho phép ứng dụng truyền và nhận dữ liệu với thiết bị khác thông qua InputStream và OutputStream.

Để có thể sử dụng những tính năng của Bluetooth ở trong ứng dụng, ta phải khai báo quyền cho phép Bluetooth. Ta cần quyền này để khởi tạo giao tiếp Bluetooth.

Và như đã đề cập đến ở phần trước, để khai báo các quyền cho phép, ta cần khai báo chúng trong file AndroidManifest.xml.



1. **Setting up Bluetooth**

Trước khi ứng dụng có thể giao tiếp thông qua Bluetooth, ta cần kiểm tra xem thiết bị hiện tại có hỗ trợ Bluetooth hay không. Trong phạm vi luận văn, do thiết bị di động được sử dụng là điện thoại Nexus 4, có hỗ trợ Bluetooth, nên không cần phải kiểm tra. Nhưng nhìn chung khi ứng dụng được phát hành phổ biến thì không nên bỏ qua bước này.

Nếu thiết bị không được hỗ trợ Bluetooth, ta đành phải vô hiệu hóa toàn bộ các tính năng Bluetooth và phải nghĩ đến các phương án thay thế khác, còn nếu thiết bị có hỗ trợ, nhưng đang trong trạng thái “disabled”, ta có thể chuyển sang trạng thái “enabled” mà không cần thoát khỏi ứng dụng. Ta có thể làm được điều này thông qua BluetoothAdapter.

* **Tạo một object BluetoothAdapter**

Để tạo được một object BluetoothAdapter, ta gọi hàm static getDefaultAdapter() của lớp BluetoothAdapter, hàm trả về một đối tượng BluetoothAdapter, đại diện cho Bluetooth Adapter mặc định của thiết bị (hay còn gọi là Bluetooth Radio). Thường chỉ có một Adapter cho một thiết bị, và ứng dụng có thể tương tác thông qua object của nó. Nêu hàm getDefaultAdapter() trả về null, nghĩa là thiết bị không hỗ trợ Bluetooth, và không thể thực hiện các tính năng của ứng dụng ta đang phát triển.



* **Enable Bluetooth**

Tiếp theo, ta phải đảm bảo rằng Bluetooth đang được enable. Ta có thể gọi hàm isEnabled() để kiểm tra, nếu hàm trả về false, nghĩa là hiện tại đang disabled. Để có thể enable Bluetooth, ta sẽ bắt đầu một Activity của hệ thống, đó là Activity hỏi người dùng có muốn enable Bluetooth hay không. Và để bắt đầu một Activity mới, ta sẽ dùng Intent như đã giới thiệu ở phần trước.

Sẽ có một hộp thoại hiện lên hỏi người dùng có muốn bật Bluetooth hay không, nếu ta chọn YES, thì hệ thống sẽ bật Bluetooth và trở lại Activity cũ của ta.

Tuy nhiên, trong phạm vi của luận văn, vì ta muốn viết một Class để định nghĩa riêng cho các Bluetooth Config, nên nó sẽ không nằm trong Activity chính, vì vậy ta sẽ viết một hàm để kiểm tra trạng thái của Bluetooth, và bật nếu đang ở trạng thái disabled.



Ở đây ta không sử dụng Intent ngay, vì ta chỉ đang định nghĩa một Class BluetoothConfig, mà ta sẽ dùng một object của Class Handler, Class này giúp giải quyết các message gửi đến đợi hệ thống xử lý. Và ở các Activity sử dụng một đối tượng của BluetoothConfig ta sẽ có hàm để xử lý các message này. Và với message BT\_REQUEST\_ENABLE, ở hàm xử lý message (hàm handleMessage(Massage) mà ta sẽ giới thiệu sau) sẽ dùng một Intent để gửi yêu cầu bật Bluetooth.

1. **Tìm kiếm thiết bị**

Vì mạch vi điều khiển STM32F4 đã kết nối sẵn với một module Bluetooth, và module này có một địa chỉ MAC cố định, nên ta chỉ cần tìm đúng module này mà không cần phải bật tính năng tìm kiếm.



Trong đó hàm getRemoteDevice(String) lấy tham số là địa chỉ MAC của thiết bị, hàm trả về một object đại diện cho module Bluetooth, và ta có thể tiến hành các kết nối, truyền nhận dữ liệu thông qua object này.

1. **Kết nối thiết bị**

Để tạo được kết nối giữa hai thiết bị với nhau, ta phải lập trình cho cả Server và Client. Vì một thiết bị phải mở một Server Socket, và thiết bị còn lại phải khởi tạo kết nối (sử dụng địa chỉ MAC của server để khởi tạo kết nối). Server và Client được coi là đã kết nối với nhau nếu cả hai đều có chung một BluetoothSocket ở cùng kênh RFCOMM. Tại thời điểm này, mỗi thiết bị có thể nhận các luồng input và output và có thể truyền dữ liệu.

Server và Client có được BluetoothSocket theo hai cách khác nhau. Server sẽ nhận được Socket khi nó chấp nhận kết nối được gửi tới. Client sẽ nhận được Socket khi nó mở một kênh RFCOMM đến server.

Một kỹ thuật lập trình được khuyên dùng là ở mỗi thiết bị, ta lập trình nó như một Server, nên mỗi thiết bị sẽ có một Socket luôn được mở và lắng nghe các kết nối gửi đến nó. Thiết bị còn lại có thể khởi tạo một kết nối với tư cách là Client.

Tuy nhiên, ở phạm vi luận văn, do ta đã biết địa chỉ MAC của module Bluetooth, mặt khác, module Bluetooth chỉ đóng vai trò truyền nhận tín hiệu, nên ta hoàn toàn có thể coi module Bluetooth là một Server, và thiết bị di động chạy hệ điều hành Android là một Client, khởi tạo kết nối đến nó.

* **Cách kết nối với vai trò một Client:**

Để có thể khởi tạo một kết nối với một thiết bị, đầu tiên ta phải có được một object BluetoothDevice đại diện cho thiết bị đó (đã trình bày ở trang trước). Sau đó ta sẽ dùng đối tượng này để tạo một BluetoothSocket và khởi tạo kết nối.

Quy trình có thể biểu diễn đơn giản như sau:

* Sử dụng object BluetoothDevice, nhận được một BluetoothSocket thông qua hàm createRfcommSocketToServiceRecord(UUID).

Hàm này sẽ khởi tạo một BluetoothSocket mà sẽ kết nối với thiết bị đại diện bởi BluetoothDevice object. Tham số UUID ở đây phải trùng với của Server khi nó mở BluetoothServerSocket của nó. Thông thường ta sẽ chọn UUID là chuỗi String:

**private** **static** **final** UUID *MY\_UUID* = UUID

.*fromString*("00001101-0000-1000-8000-00805F9B34FB");

* Khởi tạo kết nối bằng cách gọi hàm connect() của đối tượng BluetoothSocket vừa nhận được. Hàm connect() là một blocking-call (vì nếu vì lí do gì đó, kết nối bị lỗi hoặc quá 12 giây time out, hệ thống sẽ sinh ra lỗi), nên ta phải viết hàm này trong một luồng (Thread) tách biệt so với luồng của chương trình chính.

Hàm tạo Socket sử dụng trong luận văn:



Hàm trả về kiểu BluetoothSocket, tham số là đối tượng BluetoothDevice. Hàm sẽ kiểm tra phiên bản của Android hiện tại, nếu lớn hơn 10 (tương đương Android 2.3.3 trở lên), ta sẽ tạo Socket bằng hàm createInsecureRfcommSocketToServiceRecord(UUID), sử dụng hàm này vì ta chỉ sử dụng hàm createRfcommSocketToServiceRecord(UUID) trong trường hợp ta đảm bảo được độ bảo mật của kết nối, nếu một trong hai thiết bị không có khả năng nhận input và output, hoặc chỉ có khả năng hiển thị giá trị, thì không thể tạo ra một kết nối bảo mật, và Android từ 2.3.3 trở lên được khuyến cáo sử dụng hàm createInsecureRfcommSocketToServiceRecord(UUID).

Hàm tạo kết nối và tạo OutputStream:



Đầu tiên ta tạo một BluetoothSocket từ object BluetoothDevice được truyền vào từ tham số. Sau đó ta sẽ gọi hàm connect(). Nếu connect() bị lỗi thì ta sẽ gọi hàm close() để đóng nó lại.

Sau đó ta sẽ gọi hàm getOutputStream() để lấy OutputStream cho Socket. Đây là stream mà dữ liệu gửi đi từ thiết bị Android đi qua để đến module Bluetooth.

Thread độc lập để lắng nghe kết nối gửi đến thiết bị Android:



Đầu tiên ta khởi tạo đối tượng của InputStream, đây là stream để dữ liệu đi từ bên ngoài vào thiết bị Android. Ta sẽ tạo bằng cách gọi hàm getInputStream() của đối tượng Socket đã có. Ở hàm run() của Thread này (đây là hàm mô tả các thao tác của Thread), ta đặt một biến buffer là một array kiểu byte với độ dài 256, và đọc về từ InputStream thông qua hàm read(buffer), giá trị đọc về sẽ được gửi đi bằng đối tượng của Class Handler như đã trình bày ở phần trước.

Trong phạm vi của luận văn, ta chỉ truyền dữ liệu từ thiết bị Android đến module Bluetooth kết nối sẵn với vi điều khiển STM32F4, nên ta tạm thời không sử dụng Thread này, vì vậy trong hàm BT\_connect(String address, boolean listen\_Instream), tham số boolean sau ta sẽ để là false mỗi khi gọi hàm, tức là ta sẽ không sử dụng InputStream.

Để gửi được dữ liệu qua BluetoothSocket, ta sẽ định nghĩa một hàm sendData(String message) với tham số là message muốn gửi:



Do ta phải gửi từng byte theo OutputStream, ta sẽ phải tạo một array msgBuffer kiểu byte để chứa chuỗi byte của message, sau đó gửi dữ liệu thông qua hàm write(byte) của đối tượng OutputStream.

Cuối cùng ta sẽ lập trình thêm một hàm nhỏ nữa, đó là hàm sử dụng khi ta muốn ngưng sử dụng tính năng Bluetooth, để tiết kiệm năng lượng cũng như bộ nhớ, ta sẽ đóng Socket và xóa OutputStream:



Như vậy ta đã khảo sát xong những tính năng chính của Bluetooth trên hệ điều hành Android, cũng như cách tạo đối tượng BluetoothAdapter, BluetoothDevice, BluetoothSocket, các khởi tạo kết nối cũng như truyền nhận dữ liệu thông qua Bluetooth. Ở phần tiếp theo, ta sẽ khảo sát một Activity dùng để tạo một giao diện điều khiển ở chế độ Manual Mode.

1. **Manual Mode Activity**

Ở phần này, ta sẽ khảo sát một chế độ hoạt động của ứng dụng, đó là chế độ điều khiển bằng tay (Manual Mode). Và điều này đòi hỏi sự tương tác của người dùng trên thiết bị Android (chạm tay vào các Button để điều khiển xe), nên ta sẽ tạo một Activity để mô tả nó. Ta tạo file ManualMode.java để lập trình cho Activity này.

1. **Layout**

Ta sẽ tạo một layout cho Activity này và đặt tên là manual\_mode\_activity.xml. Để điều khiển xe, ta sẽ cần các 4 Button đảm nhiệm việc truyền các tín hiệu cơ bản:

* Forward: Dùng để ra lệnh đi tới
* Backward: Dùng để ra lệnh đi lùi
* Left: Dùng để ra lệnh rẽ trái
* Right: Dùng để ra lệnh rẽ phải

Ta sẽ định nghĩa bố cục của Layout là kiểu RelativeLayout, lợi điểm của RelativeLayout là nó định nghĩa vị trí của các đối tượng View một cách tương đối, ví dụ đối tượng A có thuộc tính “nằm bên phải” của đối tượng B, dẫn đến khi ứng dụng chạy trên những thiết bị có kích thước màn hình khác nhau thì Layout sẽ vẫn hiện thị đúng vị trí tương đối của các đối tượng.

Vậy ta có tổng cộng bốn Button, tương ứng cho bốn tính năng Forward, Backward, Left, Right. Để có thể canh đều thì ta sẽ định nghĩa thêm một Button nữa, gọi là Center Button, nhưng cho nó ẩn đi, vậy thì các nút tương ứng sẽ có một vị trí tương đối đối với Center Button này, tương ứng là trên, dưới, trái, phải.

Ví dụ với Forward Button:

Ở ví dụ này, ta thấy Forward Button có một thuộc tính là “layout\_above” và được gán vào ID của Center Button, cho biết là nút Forward Button sẽ nằm ở trên Center Button.

* Giới thiệu về thuộc tính ID của các đối tượng View

Như đã đề cập ở phần giới thiệu giao diện của Activity, giao diện được tạo thành bởi các đối tượng thuộc các Class như Button, TextView, EditText, … Các Class này là các Subclass của Class View. Các đối tượng muốn được sử dụng trong các file .java thì ta cần một tham chiếu đến nó. Như đã giới thiệu, những thay đổi trong các folder hoặc file sẽ được build trong file R.java, và trong file R.java này có một nơi lưu trữ các ID cho các đối tượng View tạo nên giao diện Layout. Vì vậy trong các file source code (file .java) ta có thể dễ dàng gọi các đối tượng này thông qua “id” của chúng, ta sẽ xem xét ví dụ ở trên, ta đặt ID của Forward Button là “forward”. Vì vậy trong file Activity ta có thể trỏ đến đối tượng này như sau:



Đầu tiên ta định nghĩa các đối tượng Button dùng trong chương trình, sau đó ta sẽ trỏ đến các Button định nghĩa trong Layout thông qua hàm findViewById(int id).

1. **ManualMode.java**

Sau khi đã tạo xong Layout cho Activity, ta sẽ lập trình cho Activity này thông qua file ManualMode.java.

Đầu tiên ta sẽ tạo một đối tượng của Class BluetoothConfig, đối tượng này sẽ thực hiện các hàm yêu cầu các tính năng Bluetooth: như kiểm tra trạng thái, kết nối thiết bị và truyền dữ liệu:



Sau đó tạo các đối tượng Button tương ứng với các Button định nghĩa trong Layout:

****

Như đã đề cập ở phần Life Cycle của một Activity, một Activity khi bắt đầu sẽ gọi hàm onCreate(Bundle), và ta sẽ set Layout cho Activity này thông qua hàm setContentView(int).



Tiếp đến ta gán các biến Button tương ứng với các Button định nghĩa trong Layout:



Sau đó, ta sẽ khởi tạo đối tượng BluetoothConfig và kiểm tra trạng thái của Bluetooth:



Ta khởi tạo đối tượng BluetoothConfig bằng Constructor định nghĩa trong file BluetoothConfig.java, lấy hai tham số là Context và Handler, ngụ ý là đối tượng BluetoothConfig sẽ thuộc về Activity này và đối tượng Handler của Activity này sẽ xử lý các message được gửi đi.

Hàm checkBtState() sẽ kiểm tra trạng thái Bluetooth, nếu đang disabled thì nó sẽ khởi động Activity của hệ thống để hỏi người dùng có cho phép bật Bluetooth hay không.

* **Hàm xử lý khi có tương tác của người dùng đến các Button**

Ở ngôn ngữ lập trình Java, chúng ta có khái niệm Listener, là các chương trình dùng để “lắng nghe” các sự kiện xảy ra trong quá trình tương tác của người dùng hoặc có sự thay đổi trong hệ thống, ví dụ như sự kiện Click chuột, Rê chuột, Gõ bàn phím, …

Và Android được viết trên ngôn ngữ Java, nên ta cũng sẽ có những “Listener” để nhận biết các tương tác của người dùng đến các đối tượng thông qua các sự kiện như Chạm màn hình, Click vào nút, …

Trong trường hợp Button, ta sẽ gọi hàm setOnTouchListener(OnTouchListener) để xử lý sự kiện khi người dùng chạm vào Button đó.

Ta cùng xem hàm xử lý cho Forward Button dưới đây:



Hàm setOnTouchListener nhận tham số là một đối tượng OnTouchListener, và ta phải định nghĩa hàm onTouch(View, MotionEvent), trong đó View là để chỉ đối tượng View (dùng chung cho các đối tượng mà người dùng tương tác, và MotionEvent là các sự kiện chuyển động)

Khi MotionEvent là MOVE, tương ứng khi tay ta chạm và rê màn hình, thì ta sẽ sử dụng hàm sendData(String) để gửi đi tín hiệu điều khiển đi tới, và khi MotionEvent là UP, tương ứng khi ta nhấc ngón tay lên, thì ta sẽ gửi tín hiệu để dừng xe lại.

* **Giới thiệu về các lệnh điều khiển:**

Ta sẽ gửi đi tổng cộng 10 byte tới vi điều khiển. Trong đó 5 byte đầu dùng để điều khiển bánh xe bên trái và 5 byte sau dùng để điều khiển bánh xe bên phải.

Với mỗi 5 byte, byte đầu tiên là ký tự L hoặc R, tương ứng là trái hoặc phải, để vi điều khiển có thể xác nhận lại tín hiệu điều khiển, byte tiếp theo là “+” hoặc “-” tương ứng để điều khiển quay thuận hoặc quay nghịch, trong trường hợp này “+” tương ứng là chiều làm cho xe di chuyển về phía trước và ngược lại. 3 byte tiếp theo là giá trị PWM để điều khiển động cơ.

Vậy khi điều khiển ta sẽ có các trường hợp sau:

* L+xxx R+xxx: điều khiển xe đi về phía trước, tương ứng cả hai động cơ quay thuận.
* L-xxx R+xxx: điều khiển xe rẽ trái, tương ứng động cơ trái quay ngược và động cơ phải quay thuận.
* L+xxx R-xxx: điều khiển xe rẽ phải, tương ứng động cơ trái quay thuận và động cơ phải quay ngược.
* L-xxx R-xxx: điều khiển xe đi lùi, tương ứng cả hai động cơ quay ngược.

Tiếp theo ta sẽ cùng xét Class Handler dùng trong Activity này, vì ta muốn xử lý các message theo các trường hợp tương ứng, nên ta sẽ viết lại một Class MyHandler mới mà kế thừa Class Handler.



Ở đây ta sử dụng phương thức tham chiếu đến Activity của ta theo kiểu WeakReference, đây là kiểu tham chiếu thường được dùng khi ta muốn nó được loại bỏ tự động khi ta không tham chiếu đến nó nữa, giúp tiết kiệm được năng lượng và bộ nhớ.

Và ta đang thừa kế Class Handler, ta sẽ viết lại một hàm của nó (viết lại một hàm của SuperClass được gọi là Override), đó là hàm handleMessage(Message).



Các message mà Class BluetoothConfig gửi đi gồm có:

* BT\_NOT\_AVAILABLE: message được gửi khi thiết bị không hỗ trợ Android. Khi này hệ thống sẽ gửi một dòng thông báo thông qua Class Toast, cho người dùng biết là Bluetooth không được hỗ trợ, sau đó sẽ ngưng Activity thông qua hàm finish()
* BT\_INCORRECT\_ADDRESS: message được gửi khi không thể kết nối với thiết bị do sai địa chỉ.
* BT\_REQUEST\_ENABLE: message được gửi khi ta yêu cầu bật Bluetooth, và hệ thống sẽ bắt đầu Acvtivity của hệ thống thông qua Intent.
* BT\_SOCKET\_FAILED: message được gửi khi Socket không thể tạo được.

Tương ứng với các message mà hàm handleMessage(Message) sẽ có phương thức xử lý giống như đã định nghĩa ở phía trên.

Cuối cùng, ta còn hai hàm nữa để mô tả hành vi cho Activity trong quá trình chuyển trạng thái trong Life Cycle, đó là onPause() và onResume(). Ở hàm onPause() ta chỉ đơn giản là gọi hàm BT\_onPause() của đối tượng BluetoothConfig, và kèm thêm điều kiện là Bluetooth đang được enabled, nếu không sẽ phát sinh lỗi, do hàm BT\_onPause() sẽ xóa OutputStream, mà khi đó OutputStream chưa phải là một đối tượng.

Còn trong hàm onResume(), ta cũng sẽ gọi hàm BT\_Connect(String, boolean) để kết nối lại với thiết bị.



Ta đã khảo sát xong chế độ Manual được mô tả trong file ManualMode.java. Ở chế độ này ứng dụng hoạt động và điều khiển đúng tín hiệu. Ở bên đáp ứng của vi điều khiển cũng đọc đúng tín hiệu điều khiển và hoạt động tốt.

1. **Auto Mode**

Sau khi khảo sát xong chế độ điều khiển bằng tay (Manual Mode), tiếp theo ta sẽ khảo sát đến chế độ tự động (Auto Mode).

Ta đặt tên Activity là MainActivity.java, và Layout tên là activity\_main.xml.

Vì đây là ứng dụng hướng đến mục đích là điều khiển một chiếc xe với các tính năng như:

* Nhận biết được vị trí hiện tại.
* Tìm được đường đi từ vị trí hiện tại đến vị trí muốn đến.
* Tự tính toán các góc quay giữa các điểm dừng.
* Tính toán khoảng cách và đưa ra được tín hiệu điều khiển phù hợp.

Với các tính năng mong muốn như trên, trong phạm vi của luận văn ta sẽ cần nghiên cứu sử dụng đến các hệ thống và phần cứng như sau:

* GPS
* Google Maps
* Các cảm biến của thiết bị Android

Đầu tiên ta sẽ khảo sát tổng quát những gì ta mong muốn ở những mục trên.

* GPS: GPS – Global Positioning System, là hệ thống định vị toàn cầu của Mỹ, và được sử dụng rộng rãi trên toàn thế giới. Ta sẽ sử dụng GPS để định vị được vị trí hiện tại. Ở phần sau ta sẽ khảo sát kỹ hơn về tính năng của GPS và cách thức lập trình để sử dụng GPS trên thiết bị Android.
* Google Maps: đây là một ứng dụng nổi tiếng của Google. Ta sẽ nhúng bản đồ vào Layout của Activity, và hiển thị vị trí hiện tại của ta lên bản đồ, sau đó chọn địa điểm muốn đến bằng cách chạm vào một điểm khác mà ta mong muốn. Và dựa vào Google Maps API ta sẽ lập trình để có được tập hợp các tọa độ mà ta cần đi qua, và tính toán được các tín hiệu điều khiển.
* Các cảm biến của thiết bị Android: từ các tập hợp tọa độ có được từ Google Maps API, ta sẽ tính được các góc quay cần thiết, tuy nhiên ta cần tính được góc quay hiện tại của thiết bị Android để đối chiếu với góc cần quay. Và ta sẽ thực hiện điều đó bằng cách đọc các giá trị của các cảm biến trong thiết bị Android. Các cảm biến ta sẽ sử dụng là Accelerometer và Magnetic Field. Từ giá trị của các cảm biến, ta sẽ tính toán được góc quay hiện tại của thiết bị Android.

Tiếp theo ta sẽ khảo sát chi tiết từng module ở trên.

1. **GPS – Location Based Service**

Một trong những tính năng mà thiết bị di động hiện tại đều phải tối ưu, đó chính là tính gọn nhẹ và tiện lợi. Người sử dụng cần một thiết bị cầm tay mà được tích hợp những thứ cần thiết, một trong số đó là khả năng định vị vị trí. Và trong thời đại công nghệ thông tin như hiện nay, có rất nhiều API hỗ trợ chúng ta trong việc tìm kiếm, hiển thị và đánh dấu các địa điểm.

Trong phạm vi luận văn, ta sẽ sử dụng Location Based Service – LBS để xác định vị trí hiện tại của thiết bị Android. Có nhiều kỹ thuật để thực hiện được điều này, và ta có thể chọn sử dụng một kỹ thuật nhất định, hoặc đưa ra những yêu cầu (về độ chính xác, tiết kiệm năng lượng, …) để hệ thống sẽ chọn cho chúng ta kỹ thuật nào phù hợp nhất.

* **Sử dụng Location Based Service**

Location Based Service là một từ dùng để chỉ các kỹ thuật khác nhau mà ta có thể sử dụng để xác định vị trí hiện tại. Có hai phần tử chính của LBS là:

* Location Manager: tạo đối tượng sử dụng LBS.
* Location Providers: bao gồm các kỹ thuật xác định vị trí khác nhau mà ta sử dụng để xác định vị trí hiện tại.

Sử dụng Location Manager, ta có thể:

* Tìm được vị trí hiện tại.
* Theo dõi sự di chuyển.
* Tạo được các cảnh báo tiệm cận (proximity alert) khi phát hiện chuyển động đi vào hoặc đi ra khỏi một khu vực chỉ định.
* Tìm những Location Providers được hỗ trợ.
* Theo dõi trạng thái của bộ thu GPS.

Ta có thể có được sự truy cập đến LBS thông qua Location Manager. Để truy cập được Location Manager, ta sẽ yêu cầu khởi tạo một Instance của LOCATION\_SERVICE sử dụng hàm getSystemService:



Trước khi ta có thể sử dụng LBS, ta phải khai báo các quyền cho phép vào file AndroidManifest.xml, giống như đã khai báo quyền truy cập Bluetooth ở phần trước.

Ta sẽ khai báo quyền cho phép đối với “Fine Location” và “Coarse Location” như sau:



Fine và Coarse là hai khái niệm thể hiện mức độ chính xác mà ta có thể có được khi định vị vị trí, trong đó Fine có độ chính xác cao hơn và Coarse có độ chính xác thấp hơn.

* **Lựa chọn Location Providers:**

Tùy thuộc vào thiết bị Android của mỗi người, ta có thể sử dụng những kỹ thuật khác nhau để xác định vị trí. Mỗi kỹ thuật có những khả năng khác nhau: bao gồm sự khác nhau về độ tiêu hao năng lượng, độ chính xác, và khả năng xác định thông tin về độ cao, tốc độ, …

Class LocationProvider bao gồm ba hằng số, tưng ứng trả về tên của provider của ba Location Providers:

* LocationManager.GPS\_PROVIDER
* LocationManager.NETWORK\_PROVIDER
* LocationManager.PASSIVE\_PROVIDER

GPS Provider và Passive Provider yêu cầu quyền cho phép “Fine”, trong đó Network Provider chỉ yêu cầu quyền cho phép “Coarse”.

* **Lựa chọn Provider dựa vào các điều kiện mong muốn**

Trong đa số các trường hợp, ta thường không thể chọn được đúng Provider mà mình muốn sử dụng. Khi đó, ta nên chỉ ra những yêu cầu mình mong muốn và để Android lựa chọn Provider giúp chúng ta.

Để làm được điều này, ta sẽ dùng Class Criteria để chứa các yêu cầu theo độ chính xác, độ tiêu hao năng lượng, tiêu hao tài chính, và khả năng xác định được độ cao, tốc độ, …



Sau khi định nghĩa đối tượng Criteria, ta có thể sử dụng hàm getBestProvider để trả về đối tượng Location Provider thích hợp nhất.

  
Nếu có nhiều hơn một Provider thỏa mãn Criteria, thì Provider với độ chính xác cao hơn sẽ được chọn. Nếu không có Provider nào thỏa mãn, thì đối tượng Criteria sẽ giảm dần các điều kiện theo thứ tự sau, cho đến khi tìm được Provider thỏa mãn:

* Độ tiêu hao năng lượng
* Độ chính xác của vị trí trả về
* Độ chính xác của góc phương vị, tốc độ, độ cao
* Khả năng đọc được góc phương vị, tốc độ, độ cao

Điều kiện về sự tiêu tốn cước di động không bao giờ được giảm, nếu không có Provider nào phù hợp, hàm sẽ trả về null.

Để có được một Instance của một Provider mà ta muốn đích danh, ta cần gọi hàm getProvider(String) với tham số là tên của Provider đó.



Ở phạm vi của luận văn, do phần cứng tương đối mang tính chất mô hình, và chưa thể so sánh với thực tế, nên để có thể dễ dàng trong quá trình demo, thay vì dùng GPS Provider thì ta sẽ dung Network Provider, vì GPS tuy độ chính xác cao nhưng cập nhật rất chậm do cơ chế hoạt động của GPS cần thu thập thông tin của ít nhất ba vệ tinh GPS thì mới trả kết quả vị trí về, và thực tế trong quá trình thực hiện luận văn, GPS Provider đáp ứng không như mong đợi với thời gian trễ giữa hai lần cập nhật là tương đối lâu, còn Network Provider tuy độ chính xác thấp hơn, nhưng tốc độ cập nhật rất nhanh.

* **Tìm vị trí cuối cùng được ghi nhận (Last Known Location)**

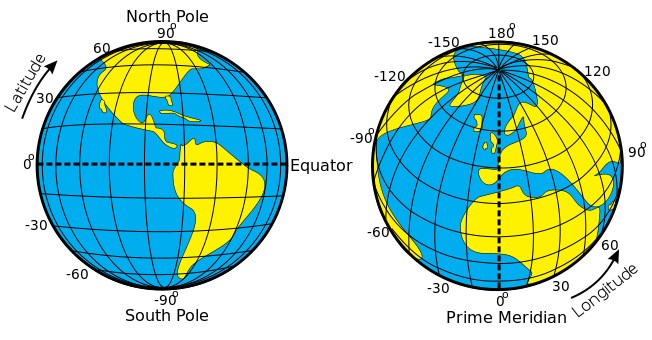
Ta có thể nhận được vị trí cuối cùng mà hệ thống ghi nhận, ta sẽ gọi hàm getLastKnownLocation, với tham số là tên của Provider được sử dụng. Ví dụ như ta muốn tìm vị trí cuối cùng được ghi nhận bằng Network Provider:



Lưu ý rằng hàm getLastKnownLocation không yêu cầu sự cập nhật mới từ Location Provider, nên vị trí nhận được có thể không còn chính xác nữa.

* **Giới thiệu về đối tượng Location, thuộc tính Latitude và Longitude**

Khi cập nhật vị trí, đối tượng Location sẽ được cập nhật, một đối tượng Location chứa bên trong rất nhiều thông tin về vị trí hiện tại, trong đó có hai thông tin quan trọng nhất, chính là Latitude và Longitude.



* Latitude: giá trị của vĩ độ, tính bằng độ, kiểu double.
* Longitude: giá trị của tung độ, tính bằng độ, kiểu double.

Với hai giá trị Latitude và Longitude mà ta thu được từ đối tượng Location, ta hoàn toàn có thể tính toán được khoảng cách giữa hai điểm, tính góc quay cần thiết, …

Sau khi đã nhận được đối tượng Location và lấy được vị trí cuối cùng mà hệ thống ghi nhận, ta sẽ tìm hiểu phương thức để cập nhật vị trí liên tục.

Đầu tiên ta sẽ thêm vào hàm onCreate() của Activity để tạo tham chiếu đến Class LocationManager, khởi tạo đối tượng Location bằng cách gán vị trí cuối cùng được hệ thống ghi nhận thông qua hàm getLastKnownLocation, và sau đó truyền vào một hàm updateWithNewLocation(Location) mà ta định nghĩa.



Để có thể theo dõi được quá trình cập nhật, ta sẽ tạo trọng Layout một đối tượng TextView để quan sát sự thay đổi của hai thông tin đã giới thiệu ở trên:



Với ID đã biết của đối tượng TextView, ta sẽ tạo một TextView tương ứng trong Activity và tham chiếu đến đối tượng tương ứng trong Layout:



Tiếp theo, ta sẽ định nghĩa hàm updateWithNewLocation(Location) mà ta đã gọi ở hàm onCreate(), mục đích của hàm updateWithNewLocation là sau khi nhận tham số là một đối tượng Location, ta sẽ lấy hai thông tin Latitude, Longitude để hiển thị lên Layout:



Ta lấy giá trị Latitude bằng cách gọi hàm getLatitude() và Longitude bằng cách gọi hàm getLongitude(), sau đó ta sẽ cho hai giá trị này vào một chuỗi String, cho hiển thị bằng hàm setText(String) của đối tượng TextView.

Ở trên, ta mới cho hiển thị thông tin của vị trí, nhưng lại chưa có phương thức để cập nhật vị trí để gửi vào hàm updateWithNewLocation(Location). Trong khi ứng dụng đang phát triển cần nhận biết được sự thay đổi của vị trí để có thể điều khiển. Để có thể cập nhật vị trí liên tục, ta sẽ gọi hàm requestLocationUpdates(provider, minTime, minDistance, listener). Trong đó provider chính là đối tượng Provider mà ta đang sử dụng, minTime là thời gian tối thiểu giữa hai lần cập nhật mà ta mong muốn, minDistance là khoảng cách tối thiểu để cập nhật, listener là phương thức lắng nghe khi có sự kiện vị trí thay đổi.

Và ta sẽ phải định nghĩa một Location Listener để dùng cho hàm requestLocationUpdates này.



Ta định nghĩa một đối tượng LocationListener, và ta Override các hàm của một đối tượng LocationListener:

* onLocationChanged: hàm xử lý khi vị trí hiện tại thay đổi. Trong hàm này ta gọi hàm updateWithNewLocation(Location) để cập nhật thông tin Latitude và Longitude ở Layout.
* onProviderDisabled: hàm xử lý khi Provider chuyển sang trạng thái disabled.
* onProviderEnabled: hàm xử lý khi Provider chuyển sang trạng thái enabled.
* onStatusChanged: hàm xử lý khi trạng thái của Provider thay đổi, ví dụ như Provider không thể lấy được thông tin vị trí, …

Trong phạm vi luận văn ta chỉ sử dụng hàm xử lý khi vị trí hiện tại thay đổi onLocationChanged(Location).

Sau khi định nghĩa một đối tượng locationListener, ta sẽ thêm hàm requestLocationUpdates vào hàm onCreate() của Activity.



Trong đó, minTime ta để là 1000 ms, tương ứng thời gian tối thiểu giữa hai lần cập nhật là 1 giây, và ta để minDistance là 0m, tương ứng sẽ không yêu cầu khoảng cách tối thiểu cho cập nhật.

Như vậy, chúng ta đã khảo sát xong những đặc trưng của GPS - Location Based Service cũng như cách thức sử dụng các tính năng định vị và cập nhật vị trí trong chương trình Android. Tiếp theo ta sẽ khảo sát một trong những module quan trọng nhất trong giải thuật, chính là Google Maps API.

1. **Google Maps API**

Google Maps là hệ thống bản đồ toàn cầu nổi tiếng của Google. Thông qua các ứng dụng trên nền tảng PC và di động, chắc hẳn chúng ta đều đã được trải nghiệm những tính năng tuyệt vời của hệ thống bản đồ này trong việc xác định các địa điểm, dò lộ trình giữa hai địa điểm, …

Vậy làm thế nào để ta có thể viết riêng cho mình một ứng dụng có thể sử dụng các tính năng của Google Maps? Google Maps API chính là câu trả lời cho chúng ta.

Google Maps API là một API cung cấp nhiều hàm cho phép chúng ta xây dựng những ứng dụng mà có thể truy cập những tính năng của Google Maps, ví dụ như:

* Hiển thị bản đồ Google Maps trên Layout của Activity.
* Tạo các điểm đánh dấu trên bản đồ và lấy được các giá trị Latitude & Longitude tương ứng của chúng.
* Tìm thông tin tuyến đường.
* Tìm lộ trình giữa hai địa điểm.
* Vẽ các đường nối giữa các điểm nằm giữa hai địa điểm.

Google Maps API hỗ trợ ta rất nhiều trong việc xác định những thông tin của địa điểm, lộ trình, tuy nhiên việc sử dụng được Google Maps API cũng đòi hỏi một vài tùy chỉnh cấu hình và các thủ tục khác. Mặt khác, bản thân Google Maps API sẽ không hoàn toàn giúp ta hoàn thành các công việc trên, ta vẫn phải định nghĩa thêm nhiều hàm giải thuật khác nhau và cùng kết hợp thì mới có thể giải quyết được công việc nêu trên. Quy trình bao gồm:

* Tùy chỉnh để có thể sử dụng Google Maps API.
* Dựa vào API ta sẽ nhận được một file JSON chứa thông tin lộ trình của hai địa điểm mong muốn.
* Viết các Class và hàm để quy đổi file JSON về thành những chuỗi giá trị tương ứng với các địa điểm trung gian.
* Vẽ lộ trình đi qua các địa điểm nằm trong chuỗi ở trên.
* Cùng với Location Based Service và kết quả đọc được từ các Sensor, ta sẽ đưa ra các giải thuật điều khiển phù hợp.

Do là module quan trọng nhất, nên ta sẽ đi chi tiết vào từng phần nhỏ, và sẽ xét đến giải thuật điều khiển sau cùng. Bây giờ ta sẽ khảo sát các bước tùy chỉnh cần thiết để có thể sử dụng Google Maps API.

1. **Tùy chỉnh để sử dụng được Google Maps API**

* Đầu tiên, ta cần có Android SDK, điều này là dĩ nhiên và ta đã hoàn thành ở phần I khi giới thiệu hệ điều hành Android.
* Sau đó ta cần cài đặt Google Play Service SDK. Như đã nói ở phần trước, Google Play Service SDK chứa rất nhiều bộ API của Google, trong đó có Google Maps API, và ta cũng đã làm xong công đoạn này ở phần I.
* Lấy API Key, đây là một Key cần thiết để ứng dụng của ta có thể truy cập được các tính năng của Google Maps.
* Khai báo các quyền cho phép trong file AndroidManifest.xml.
* Thêm bản đồ vào trong Activity của ứng dụng.

Sau khi đã cài đặt Android SDK và Google Play Service SDK, ta cần khai báo việc sử dụng Google Play Service SDK trong file AndroidManifest.xml, khai báo này đặt bên trong phần tử <application>

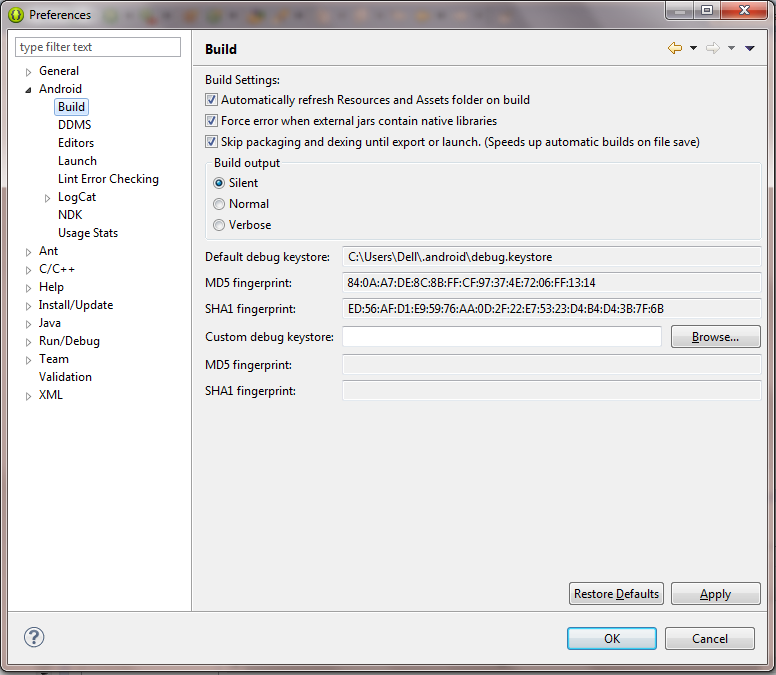


* **Lấy chứng chỉ Android (Android Certificate) và Google Maps API Key**

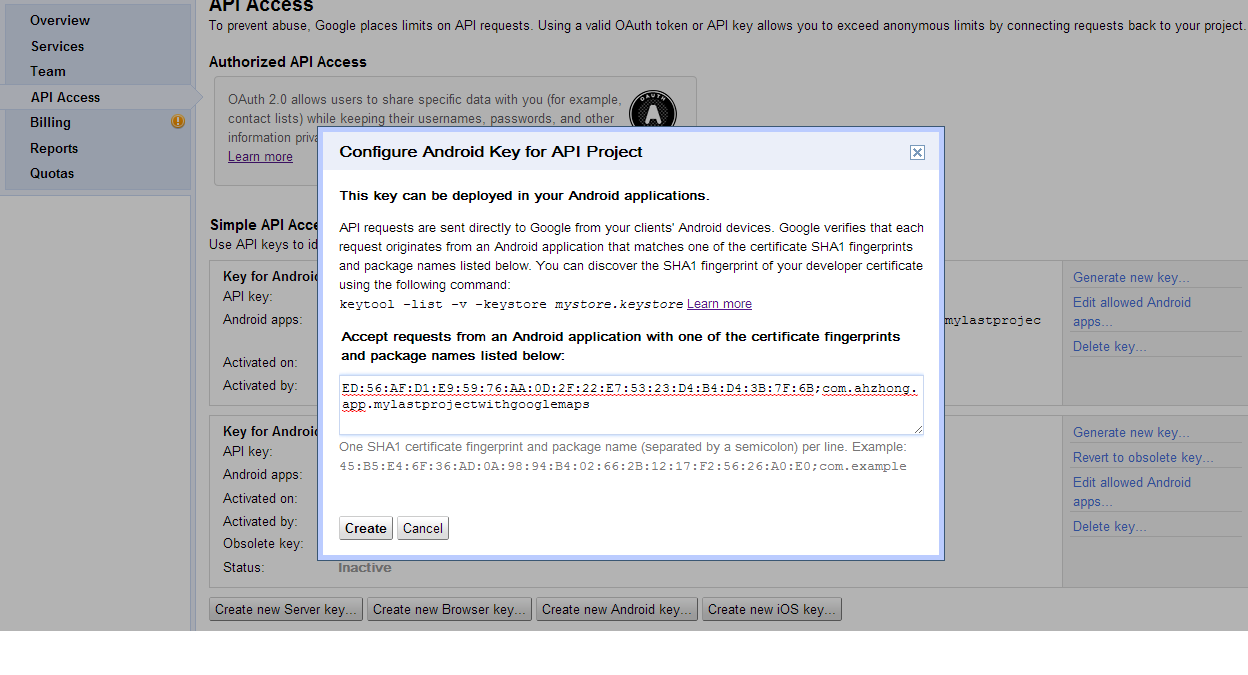
Phiên bản API mà ta đang sử dụng là phiên bản version 2, ta cần lưu ý điều này vì hiện tại Google không còn hỗ trợ version 1 nữa. Vì vậy Key của version 1 sẽ không thể sử dụng lại cho version 2.

Để có thể truy cập được Google Maps Server bằng Maps API, ta phải có khai báo API Key trong ứng dụng. Key này hoàn toàn miễn phí và có thể sử dụng cho bất kỳ ứng dụng nào muốn gọi Maps API và không giới hạn số lượng người dùng. Ta sẽ nhận API Key này từ Google API Console bằng cách cung cấp chứng chỉ Android và tên của package của ứng dụng.

Để có được chứng chỉ Android, trước đây ta phải chạy các lệnh trong command promt để hiển thị, tuy nhiên hiện tại, với IDE Eclipse ta dễ dàng có được thông tin này bằng cách: Vào Windows -> Preferences -> Android -> Build, tại đây ta có được thông tin của hai dạng mã hóa của chứng chỉ Android, bao gồm mã hóa MD5 và SHA-1. Thông tin mã hóa SHA-1 chính là chuỗi ta cần.



Sau khi lấy được mã hóa SHA-1 của chứng chỉ Android, ta sẽ truy cập vào Google API Console. Chọn API Access -> Create New Android Key. Ở cửa sổ hiện lên ta điền mã hóa SHA-1 ở trên, theo sau bởi dấu “;” và tiếp theo là tên package của project.



Sau khi lấy xong được API Key, những công đoạn config còn lại tương đối đơn giản, đó là khai báo các quyền cho phép trong file AndroidManifest.xml, chỉnh sửa file Layout của Activity và lập trình file mã nguồn của Activity để giao tiếp với Google Maps.

* **Khai báo API Key vào ứng dụng**

Ta sẽ khai báo API Key vào trong file AndroidManifest.xml. Sau đó, Maps API sẽ đọc API Key này và gửi đến Google Maps Server, sau đó sẽ xác nhận rằng ta có quyền truy cập dữ liệu Google Maps.

Ở file AndroidManifest.xml, ta thêm phần tử dưới đây bên trong phần tử <application>, bằng cách thêm vào ngay phía trên của tag </application>



Trong đó ở thuộc tính value ta điền API Key mà ta đã nhận được ở các bước như phía trên.

Sau đó ta lưu lại file AndroidManifest.xml và build lại ứng dụng.

* **Khai báo các thiết lập trong file AndroidManifest.xml**

Sau khi đã hoàn thành các bước ở trên (tham chiếu đến Google Play Service, khai báo API Key cho ứng dụng), ta cần làm thêm những công đoạn sau:

* Khai báo các quyền cho phép (permissions) mà cho phép ứng dụng được phép truy cập các tính năng hệ thống của Android và Google Maps Server.
* Thông báo cho biết ứng dụng yêu cầu sử dụng OpenGL ES version 2. Các service bên ngoài có thể phát hiện điều này và hành xử tương ứng.

Khai báo các quyền cho phép theo cú pháp sau:



Ta cần khai báo các quyền sau:

* android.permission.INTERNET: sử dụng để download Maps từ Google Maps Server.
* android.permission.ACCESS\_NETWORK\_STATE: cho phép API kiểm tra trạng thái kết nối để quyết định có thể download dữ liệu hay không.
* com.google.android.providers.gsf.permission.READ\_GSERVICES: cho phép API truy cập Google Web-based Service.
* android.permission.WRITE\_EXTERNAL\_STORAGE: cho phép API lưu tạm (cache) dữ liệu bản đồ ở bộ nhớ ngoài của thiết bị.

Sau khi khai báo các quyền cho phép, ta cần khai báo yêu cầu sử dụng OpenGL ES version 2 để render bản đồ. Nếu không có OpenGL ES v2, bản đồ sẽ không xuất hiện trong ứng dụng. Ta sẽ thêm phần tử sau bên trong phần tử <manifest>:



Sau khi hoàn thành các bước tùy chỉnh, ta sẽ tiến hành hiển thị bản đồ trên Activity. Để làm được điều này, ta sẽ cần tùy chỉnh Layout và file mã nguồn của Activity.

* **Thêm bản đồ vào ứng dụng**

Cách đơn giản nhất để kiểm tra những bước tùy chỉnh ở trên đã hoàn thành đầy đủ chưa, và cũng là để sử dụng cho các thuật toán sau này, ta sẽ tiến hành thêm bản đồ vào ứng dụng. Như đã đề cập ở trang trước, ta sẽ phải thay đổi hai file: một là file Layout định nghĩa giao diện của Activity và file mã nguồn của Activity.

Như đã đề cập ở phần AutoMode, file Layout ta đặt tên là activity\_main.xml, file mã nguồn tên là MainActivity.java. Vì ta muốn Activity chinh của ứng dụng là AutoMode, và ta có thể chuyển sang ManualMode bằng một Intent.

Ở file activity\_main.xml, ta thêm một phần tử <fragment> vào layout, phần tử này sẽ là nơi hiển thị bản đồ trên giao diện của ứng dụng.



Sau đó, để bản đồ hiển thị trên giao diện, ta cần tùy chỉnh file MainActivity.java, bằng cách đặt giao diện là file activity\_main.xml.



Sau khi build chương trình, ta thấy bản đồ hiển thị và thao tác tốt như zoom, xoay, …

Tiếp theo ta sẽ khảo sát cách thêm một điểm đánh dấu lên bản đồ, sau này điểm đánh dấu này sẽ là địa điểm đến của chúng ta.

* **Thêm một điểm đánh dấu trên bản đồ**

Điểm đánh dấu, hay còn gọi là các Marker, là những điểm chỉ một vị trí trên bản đồ. Từ các Marker, ta có thể nhận diện được thông tin của vị trí đó (Latitude & Longitude). Các Marker sử dụng một icon đặc trưng của Google, thường thấy trong úng dụng Google Maps, và API cho phép chúng ta có thể thay đổi icon này.



Các Marker được thiết kế để có thể tương tác được với người dùng. Mặc định nó có sự kiện Click, khi người dùng Click vào nó sẽ hiện thông tin. Và ta có thể chỉnh sửa được thông tin mà nó truyền tải. Ngoài ra ta còn có thể lập trình để Marker có thể drag được (bằng cách set thuộc tính draggable là true).

Để có thể thêm một Marker vào bản đồ, trước tiên ta sẽ phải tạo một đối tượng bản đồ để tham chiếu tới bản đồ ở trong Layout.



Đầu tiên ta tạo đối tượng GoogleMap, sau đó ta tạo đối tượng SupportMapFragment để tham chiếu đến Fragment sử dụng trong Layout, và sau đó ta nhận được một đối tượng GoogleMap bằng hàm getMap() của Fragment vừa tạo.

Để tạo được một Marker, ta sẽ sử dụng Class MarkerOptions. Ta sẽ tạo một đối tượng MarkerOptions:



Sau khi tạo được một đối tượng của MarkerOptions rồi, ta sẽ tùy chỉnh tọa độ của Marker này để có thể hiển thị lên bản đổ:



Ta có một Class là LatLng, dùng để tạo những đối tượng chứa thông tin về Latitude và Longitude của vị trí. Ở đây ta sử dụng để tạo ra đối tượng LatLng, và đưa nó vào làm tham số của hàm position(LatLng) để xác định vị trí cho Marker, sau đó ta tùy chỉnh thông tin của Marker (đây là thông tin sẽ hiện lên khi ta Click vào Marker), ta sẽ cho hiện lên hai thông tin là Latitude và Longitude.

Cuối cùng, ta thêm Marker này vào bản đồ thông qua hàm addMarker(MarkerOptions) của đối tượng GoogleMap.

Đoạn code trên chỉ là ví dụ, vì rõ ràng ta thấy đối tượng LatLng vẫn đang là null, ta không thể truyền một giá trị null vào hàm position(). Sau này khi vào giải thuật ta sẽ nói rõ hơn điều này.

Trong các thông tin của Marker, position là thông tin duy nhất bắt buộc phải có, những thông tin còn lại như Anchor, Title, Icon, Visible, Draggable có thể tùy chỉnh hay không tùy chỉnh cũng được.

Ta sẽ xét một ví dụ để tùy chỉnh màu sắc của Marker.

Để tùy chỉnh màu sắc cho Marker, ta sẽ truyền một đối tượng BitmapDescriptor vào hàm icon() của Marker, để có được đối tượng BitmapDescriptor, ta sẽ dùng giá trị trả về của hàm BitmapDescriptorFactory để chỉnh màu.

Trong đoạn code ở trên, ta dùng một đối tượng LatLng đặt tên là MELBOURNE để lữu giữ thông tin Latitude và Longitude của thành phố Melbourne, sau đó ta add Marker này với vị trí là đối tượng LatLng đó, cùng màu sắc của icon thay đổi thành màu Azure (tên của một màu xanh dương).

Như vậy ta đã khảo sát về các tính năng của một Marker trên bản đồ Google Map cũng như cách thức thêm Marker này trên bản đồ cũng như có thể tùy chỉnh các thuộc tính của nó.

Phần tiếp theo ta sẽ khảo sát cách thức để trích ra được thông tin của các địa điểm trung gian nằm giữa hai địa điểm, từ đó tiến gần hơn tới công đoạn vẽ lộ trình đi qua hai điểm và thuật toán điều khiển.

1. **Từ Google Maps API, nhận được file JSON chứa thông tin về lộ trình giữa hai địa điểm**

Để có thể thực hiện được giải thuật điều khiển, công việc ta cần làm là phải chia nhỏ lộ trình giữa hai địa điểm thành nhiều đoạn nhỏ, vì ta chỉ có thể di chuyển trên những tuyến đường. Và những địa điểm trung gian này sẽ giúp ta bám vào các tuyến đường, cũng như tính toán các góc quay chính xác hơn.

Tương tự như khi ta sử dụng ứng dụng Google Maps của Google, khi muốn xem lộ trình giữa hai địa điểm, đầu tiên Google sẽ tính toán để cho ra được những địa điểm trung gian giữa địa điểm đầu và địa điểm cuối, sau đó sẽ vẽ các đoạn thẳng nối các địa điểm này lại với nhau, cuối cùng ta được một đường kỷ hà nối giữa hai địa điểm và bám theo các tuyến đường.

Và Google cũng cho phép chúng ta làm được điều tương tự như vậy, ta cần có thông số Latitude & Longitude của địa điểm đầu và địa điểm cuối, Google Maps API sẽ cung cấp cho chúng ta thông tin về lộ trình, những địa điểm trung gian cần thiết cũng như nhiều thông tin hữu ích khác (khoảng cách, thời gian di chuyển, …)

Và cách thức để thực hiện việc này, ta sẽ phải thông qua URL, chứ không phải thông qua hàm. Vì Google chỉ cung cấp thông tin này thông qua URL. Cú pháp URL này như sau:

[https://maps.googleapis.com/maps/api/directions/json?origin=originLat, originLng&destination=destLat,destLng&sensor=false](https://maps.googleapis.com/maps/api/directions/json?origin=originLat,%20originLng&destination=destLat,destLng&sensor=false)

Ta sẽ khảo sát từng phần của URL này, đầu tiên là “json?”, Google có hai phương thức để gửi thông tin giữa hai địa điểm, một là JSON và một là XML. Cả hai đều là những định dạng dùng để truyền tải thông tin qua các giao thức mạng. Như ta đã biết về định dạng XML, gồm các phần tử chứa thông tin được chứa trong các Tag. Vì vậy khi nhận được file XML, dựa vào tên các Tag ta có thể lọc trích ra được những phần tử phù hợp. File JSON cũng có cách thức hoạt động gần giống như vậy, và Android có hỗ trợ một bộ Parser dành cho file JSON, nên ta chỉ cần chỉnh sửa lại để lấy được các thông tin ta cần.

Tiếp theo, origin=originLat,originLng, đây là thông tin về Latitude & Longitude của địa điểm đầu, ta thay giá trị vào originLat và originLng.

Tương tự là destination=destLat,destLng, đây là thông tin về Latitude & Longitude của địa điểm đến, ta thay giá trị vào destLat và destLng.  
Ví dụ của một URL hoàn chỉnh:

<https://maps.googleapis.com/maps/api/directions/json?origin=10.815,106.674&destination=10.822,106.911&sensor=false>

* **Giới thiệu định dạng file JSON**

JSON – JavaScript Object Notation, là một format sử dụng trong truyền nhận dữ liệu. Lợi điểm của nó là gần với ngôn ngữ con người, ta có thể dễ dàng đọc và viết. Và việc Parse cũng như Generate các file này tương đối đơn giản đối với máy tính. Tuy JSON là một format độc lập nhưng sử dụng các quy ước quen thuộc đối với các lập trình viên C, C++, C#, Java, Perl, Python, … Nên định dạng JSON là một ngôn ngữ chuyển đổi dữ liệu lý tưởng.

JSON được build dựa trên hai kiểu cấu trúc:

* Một tập hợp các cặp name/value. Trong nhiều ngôn ngữ, đây được coi là một **object**, hashtable, …
* Một danh sách có thứ tự các value. Trong nhiều ngôn ngữ, đây được coi là một **array**, list, vector, …

Tổng kết lại, trong cấu trúc của định dạng JSON, các kiểu có những dạng như sau:

* Một object là một tập hợp các cặp name/value không tính thứ tự, một object bắt đầu bằng { và kết thúc bằng }. Mỗi name được theo sau bởi : và các cặp name/value được phân tách bởi dấu ,



* Một array là một tập hợp các value có tính thứ tự. Một array bắt đầu bằng [ và kết thúc bằng ]. Các giá trị value được phân tách bởi ,



* Một value có thể là một chuỗi string (nằm trong “”), hoặc một number, hoặc true/false, hoặc null, hoặc có thể là object/array. Những cấu trúc này có thể lồng vào nhau.
* Một string là một chuỗi chứa 0 hoặc nhiều ký tự Unicode, được đặt giữa hai dấu nháy kép “”. Kiểu string này rất giống với string của C hay Java.
* Kiểu cuối cùng là kiểu number, rất giống với kiểu number của C hay Java, ngoại trừ nó không có định dạng octal và hexadecimal.

Như vậy ta đã hoàn thành xong phần khảo sát định dạng của một file JSON, và qua đó ta thấy được lợi điểm rất lớn của định dạng JSON là cách định nghĩa Object, Array, String, Number của nó rất gần với ngôn ngữ Java, ngôn ngữ mà ta sử dụng để lập trình Android. Nên việc chuyển đổi các giá trị sẽ trở nên rất dễ dàng, hơn nữa Android cũng hỗ trợ ta một bộ Parser để hỗ trợ trong việc chuyển đổi dữ liệu.

Tiếp theo chúng ta sẽ tìm hiểu cách thức download file JSON từ Google Maps Server và sau đó sẽ khảo sát những hàm cần thiết, sử dụng bộ Parser của Android cung cấp để lấy thông tin của các địa điểm trung gian giữa điểm đầu và điểm cuối.

* **Download file JSON từ URL**

Đầu tiên ta sẽ viết một hàm getDirectionUrl(LatLng, LatLng) với tham số là đối tượng LatLng của điểm đầu và điểm đích, sau đó ta sẽ thành lập URL mà ta đã đề cập ở trên. Ta sẽ thiết lập file output là JSON, thay các giá trị Latitude và Longitude vào đúng vị trí, và kết hợp lại thành một chuỗi String duy nhất.



Sau khi nhận được một String thể hiện URL nơi mà ta sẽ lấy file JSON, ta sẽ tiến hành mở URL này và download file JSON đó thành một String dùng làm tham số cho các hàm Parser.

Ta sẽ viết một hàm để làm điều này, đó là hàm downloadUrl(String) với tham số là String nhận được từ hàm ở trên. Để có thể download dữ liệu từ một URL, ta sẽ phải tạo một đối tượng InputStream, một đối tượng HttpURLConnection và một đối tượng URL.



Ở hàm trên, ta khởi tạo đối tượng HttpURLConnection bằng hàm openConnection() của đối tượng URL, và sau đó gọi hàm connect() của nó để kết nối đến URL, và lấy InputStream bằng hàm getInputStream(). Sau đó ta tạo một đối tượng BufferedReader để đọc về InputStream. Tiếp đến ta dùng một đối tượng StringBuffer để xây dựng chuỗi String để trả về.

Sự khác nhau giữa String và StringBuffer trong Java:

* Trong Java, String là một Class mà ta sẽ không thể thay đổi giá trị tham chiếu của nó. Ví dụ như ban đầu ta gán String string = “a”, thì đối tượng này sẽ trỏ đến một ô nhớ có giá trị “a”, và giá trị “a” này không thể thay đổi được, nếu ta gán tiếp string = “b”, thì nó sẽ tham chiếu sang ô nhớ khác có giá trị “b”, còn ô nhớ cũ sẽ bị hệ thống loại bỏ. Như vậy ta sẽ không thể thay đổi nội dung của một giá trị tham chiếu.
* Và Java có cung cấp cho ta hai Class gần như tương đương để hoàn thành việc kể trên, đó là StringBuilder và StringBuffer. Đối tượng của hai Class này có thể chỉnh sửa, thêm bớt hoặc xóa những giá trị mà chúng tham chiếu đến. Vì vậy trong trường hợp này, ta sẽ xây dựng chuỗi bằng Class StringBuffer và sau khi hoàn thành ta sẽ chuyển đổi thành đối tượng String.

Ta sẽ đọc từng hàng của buffer và thêm vào đối tượng StringBuffer bằng hàm append(String). Sau khi hoàn tất ta sẽ chuyển đổi đối tượng StringBuffer thành String bằng hàm toString().

Sau khi hoàn tất, ta cần phải đóng InputStream và ngắt kết nối tới URL để tiết kiệm tài nguyên cho thiết bị.

Như vậy ta đã viết xong hàm có chức năng download nội dung của URL (file định dạng JSON) thành một đối tượng String. Tuy nhiên, ta không thể sử dụng hàm này trong Thread chính của chương trình được. Vì bắt đầu từ Android phiên bản 3.0 trở về sau, các quy trình đồng bộ (ví dụ như download file, …) sẽ không được chạy trong Thread chính nữa. Các quy trình đồng bộ là những quy trình sẽ chiếm quyền kiểm soát cho đến khi nó hoàn thành, nên nó sẽ làm tê liệt giao diện người dùng trong lúc nó đang thi hành. Nên từ Android 3.0 trở đi, tất cả những quy trình này phải được chạy trong một Class AsyncTask, nghĩa là các quy trình không đồng bộ, khi đó những quy trình sẽ được chạy trong một Thread riêng biệt và trả kết quả về Thread chính của chúng ta, và ta không cần giải quyết các vấn đề cạnh tranh tài nguyên phức tạp giữa các Thread.

* **Giới thiệu khai báo AsyncTask**

Để sử dụng được AsyncTask, ta phải tạo một Class mới để làm SubClass của nó. Và Class này ít nhất phải Override hàm doInBackground(Params …) và thường ta sẽ Override cả hàm onPostExecute(Result).

Có ba kiểu dữ liệu của một AsyncTask:

* Params: kiểu của các tham số để truyền vào làm tham số của AsyncTask.
* Progress: kiểu của quá trình được xuất ra trong quá trình tính toán.
* Result: kiểu của kết quả của quá trình tính toán.

Không phải lúc nào ta cũng sử dụng hết cả ba tham số này, khi có thông số không cần thiết, ta sẽ để là void.

Khi một AsyncTask được thực thi, nó sẽ đi qua bốn bước:

* onPreExecute(), được gọi trong Thread chính trước khi được thực thi, chỉ dùng để set up.
* doInBackground(Params…): được gọi ngay sau khi onPreExecute() kết thúc. Đây là bước thực hiện các quá trình tính toán mà mất nhiều thời gian. Tham số của AsyncTask được truyền vào hàm này. Kết quả của tính toán phải được trả về trong bước này và sẽ được truyền đến bước cuối cùng. Bước này có thể gọi hàm publishProgress(Progress…) để xuất các giá trị giữa của quá trình, những giá trị này có thể xuất ra Thread chính bằng bước onProgressUpdate(Progress…).
* onProgressUpdate(Progress…): được gọi sau khi gọi hàm publishProgress(Progress…). Thời gian thực thi không cố định. Hàm này dùng để xuất các giá trị trung gian khi việc tính toán vẫn đang thực thi. Có thể thấy ứng dụng của hàm này trong việc hiển thị các thanh thông báo tiến độ, …
* onPostExecute(Result): được gọi sau khi việc tính toán hoàn tất. Kết quả của quá trình tính toán được truyền vào hàm này.

Vậy ta sẽ tạo một SubClass của AsyncTask, ta sẽ truyền vào URL kiểu String, và nhận kết quả về là một String chứa nội dung file JSON, còn kiểu biến của các quá trình ta không quan tâm, nên ta sẽ khai báo có dạng sau:



Ta truyền tham số lần lượt là String, Void và String. Ta để ý là ta chỉ được gọi AsyncTask duy nhất một lần trong một lần chạy ứng dụng, vì vậy các hàm lấy tham số là một Array. Và trong chương trình chính ta sẽ tạo một đối tượng Class DownloadTask và truyền tham số cho nó là URL.



Trong đó hàm execute(Params…) sẽ khởi động một AsyncTask. Nếu ta có nhiều URL cần xử lý thì ta có thể truyền vào nhiều URL

downloadTask.execute(url1, url2, url3);

Như vậy đến đây ta đã lấy được một file định dạng JSON từ URL của Google, sau đó tạo một đối tượng String tham chiếu đến nội dung của file JSON đó. Tiếp đến ta sẽ khảo sát về những thông tin mà file JSON chứa, sau đó sẽ tiến hành viết các hàm phù hợp để lấy các thông tin mà ta cần từ file JSON đó.

* **Ví dụ về một file JSON trả về từ URL của Google:**

Khi ta tạo một URL, trong đó tham số là tọa độ Latitude & Longitude của điểm đầu và điểm đến, Google sẽ tìm thông tin của hai địa điểm đó, và tính toán cho ta rất nhiều thông tin quan trọng, như lộ trình để đi từ điểm này đến điểm khác, khoảng cách của lộ trình đó, thời gian ước tính để đi hết lộ trình, và quan trọng là thông tin về các địa điểm trung gian mà ta cần phải đi qua.

Ví dụ một đoạn của file JSON:



Như đã giới thiệu ở phần định dạng JSON, ta thấy file JSON mà ta nhận được từ URL thực chất là một JSON Object, vì nó bắt đầu bằng { và kết thúc là }. Trong Object này, ta cần lưu ý đến các giá trị sau:

* Routes: đây là tên của một giá trị, và ta dễ thấy giá trị của nó là một Array do bắt đầu bằng [ và đương nhiên sẽ kết thúc bằng ].
* Legs: nằm trong một Object bên trong giá trị Array của Routes, ta thấy giá trị của Legs cũng là một Array vì nó bắt đầu bằng [ và đương nhiên sẽ kết thúc bằng ].
* Steps: nằm trong một Object bên trong của giá trị Array của Legs, ta thấy giá trị của Steps cũng là một Array vì nó bắt đầu bằng [ và đương nhiên sẽ kết thúc bằng ].

Routes thể hiện lộ trình giữa hai địa điểm, nếu Google tìm được nhiều lộ trình cho hai địa điểm này, ta sẽ có nhiều giá trị Routes, tuy nhiên, khi khởi tạo URL, nếu ta không thêm thuộc tính Alternative thì mặc định Google sẽ chỉ tìm duy nhất một tuyến đường đầu tiên mà nó tính toán được. Nếu ta chỉ có hai địa điểm thì ta sẽ chỉ có một Legs tương ứng, nếu ta có nhiều địa điểm (hơn hai) cần tìm lộ trình thì ta sẽ có nhiều Legs, trong phạm vi luận văn, ta sẽ chỉ có một Routes và một Legs. Tuy nhiên, một Legs sẽ có một Steps, và Steps là tên của Array chứa những thông tin về các đoạn trung gian của các lộ trình, nên Array đó sẽ chứa rất nhiều Object. Và để thu thập được thông tin của các điểm trung gian, ta sẽ tìm ở trong các Object của Array của Steps.

Ta để ý rằng ở Legs và Steps ta đều có thông tin của điểm bắt đầu (start\_location), điểm kết thúc (end\_location), được thể hiện thông qua hai giá trị Latitude và Longitude. Ngoài ra còn có các thông tin về khoảng cách, thời gian ước tính, hướng dẫn cách di chuyển, … Và có một giá trị lưu giữ các địa điểm trung gian, đó chính là giá trị của “polyline”, ở file trên cụ thể sẽ là:



Như vậy, ta đã có được giải thuật để có thể lấy được thông tin của giá trị “points” trong Object của “polyline”. Giá trị đó thực ra là các tập hợp các giá trị Latitude và Longitude của các địa điểm tạo nên Steps đó, và đã được mã hóa, ta sẽ tìm hiểu cách mã hóa cũng như giải mã ở phần sau. Bây giờ ta sẽ xây dụng giải thuật để rút trích được giá trị mã hóa này từ một đối tượng String tham chiếu nội dung của file JSON mà ta có được ở phần trước.

* **Giải thuật trích giá trị mã hóa của các địa điểm trung gian.**

Tạo một đối tượng JSON Object bằng cách truyền vào giá trị của đối tượng String tham chiếu đến nội dung của file JSON:



Sau đó từ Object trên, ta sẽ lấy giá trị Array của Routes, ta sẽ dùng hàm getJSONArray(String) với tham số là tên của giá trị, chính là “routes”:



Sau khi có được Array này rồi, như đã đề cập ở phần trên, giá trị đầu tiên của Array này là một Object, trong Object này lại có một cặp Name/Value của Legs, vậy nên ta sẽ lấy được giá trị Array của Legs như sau:



Và tương tự cho việc lấy giá trị Array của Steps:

  
Như vậy là ta đã lấy được Array của Steps, tiếp đến ta sẽ dùng một vòng lặp để duyệt qua tất cả các Object của Array của Steps, ở mỗi Object đó, ta sẽ tìm “polyline”, sau đó ta lấy Object của “polyline” và tìm tiếp “points”, giá trị thu được chính là thông tin mã hóa mà ta cần:



Trong phạm vi luận văn, ta sẽ chỉ có một Routes, một Legs và một Steps, tuy nhiên để tổng quát, ta vẫn nên dùng vòng lặp cho việc lấy Array của Routes, Legs và Steps để không bỏ sót trong các trường hợp tổng quát.

Với thuật toán như trên, sẽ tốt hơn nếu ta viết chúng ở một Class riêng biệt với Activity chính, vì vậy ta sẽ tạo một Class mới với tên là DirectionsJSONParser, và viết thuật toán trên vào một hàm là parse(JSONObject) với tham số là một JSON Object.

Vậy ta sẽ cần trả về kết quả gì từ hàm này? Ta sẽ cần một Array chứa một Array chứa một bảng tra chứa giá trị của Latitude và Longitude. Cụ thể như sau:

Sau khi nhận được một giá trị mã hóa của “points” như ở trên, ta sẽ giải mã, và có được tập hợp các đối tượng LatLng, và ta sẽ lưu các giá trị này tương ứng vào một bảng tra, gọi là HashMap, như sau:

|  |  |
| --- | --- |
| Latitude | Longitude |
| Value (String) | Value (String) |

Và ứng với một Legs, ta sẽ có một ArrayList chứa tất cả các HashMap của Legs đó, tức là toàn bộ các điểm trung gian của toàn bộ các Object trong Steps.

Sau đó ta sẽ dùng một ArrayList lớn hơn dùng để chứa các ArrayList của Legs này.

Trong phạm vi luận văn, ta chỉ có một Routes và một Legs, tuy nhiên, trong trường hợp tổng quát, số Routes và Legs có thể nhiều hơn một, vì vậy ta nên lưu kết quả ở dạng như trên để không phải viết lại hàm sau này.

Tiếp đến ta sẽ tìm hiểu về chuẩn mã hóa các thông tin Latitude và Longitude.

* **Thuật toán mã hóa thông tin polyline**

Quá trình mã hóa chuyển đổi các giá trị binary thành một chuỗi các giá trị ASCII, và các giá trị được mã hóa sẽ được cộng với 63 (ký tự ?) trước khi chuyển đổi thành ký tự ASCII

Quy trình mã hóa:

* Lấy giá trị ban đầu: ví dụ -179.9832104
* Nhân giá trị này với 1e5 và làm tròn kết quả: -17998321
* Đổi sang số nhị phân, lưu ý là đối với số âm thì ta sẽ dùng số bù hai của nó:

00000001 00010010 10100001 11110001  
11111110 11101101 01011110 00001110  
11111110 11101101 01011110 00001111

* Shift giá trị nhị phân sang trái 1 bit: 1111110 11101101 01011110 000011110
* Nếu giá trị thập phân âm, ta sẽ đảo giá trị này:

00000010 00100101 01000011 11100001

* Ta sẽ phân chia giá trị này theo từng nhóm 5 bit (bắt đầu từ bên phải):

00001 00010 01010 10000 11111 00001

* Đặt các nhóm 5 bit này theo thứ tự ngược:

00001 11111 10000 01010 00010 00001

* OR từng nhóm với 0x20 ngoại trừ nhóm cuối cùng bên phải, và ta sẽ nhận được các nhóm 6 bit (do 0x20 có 2 bit cuối là 0):

100001 111111 110000 101010 100010 000001

* Chuyển đổi các giá trị thành số thập phân:

33 63 48 42 34 1

* Cộng 63 vào các giá trị:

96 126 111 106 97 64

* Chuyển đổi từng giá trị ra mã ASCII:

~oia@

Ta xét thêm một ví dụ như sau:

Giả sử ta cần mã hóa ba địa điểm: (38.5, -120.2), (40.7, -120.95), (43.252, -126.453)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Latitude** | **Longitude** | **Latitude in E5** | **Longitude in E5** | **Change In Latitude** | **Change In Longitude** |
| 38.5 | -120.2 | 3850000 | -12020000 | +3850000 | -12020000 |
| 40.7 | -120.95 | 4070000 | -12095000 | +220000 | -75000 |
| 43.252 | -126.453 | 4325200 | -12645300 | +255200 | -550300 |

|  |  |  |
| --- | --- | --- |
| **Encoded Latitude** | **Encoded Longitude** | **Encoded Point** |
| \_p~iF | ~ps|U | \_p~iF~ps|U |
| \_ulL | nnqC | \_ulLnnqC |
| \_mqN | vxq`@ | \_mqNvxq`@ |

Vậy chuỗi polyline mà ta mã hóa được là \_p~iF~ps|U\_ulLnnqC\_mqNvxq`@

Dựa vào cách mã hóa như trên, ta hoàn toàn có thể viết hàm giải mã chuỗi polyline ta nhận được bằng cách làm ngược lại các bước mã hóa như ở trên, và ta sẽ nhận được một chuỗi các đối tượng LatLng, sau đó ta sẽ lưu chúng vào ArrayList đã đề cập ở phần trên.

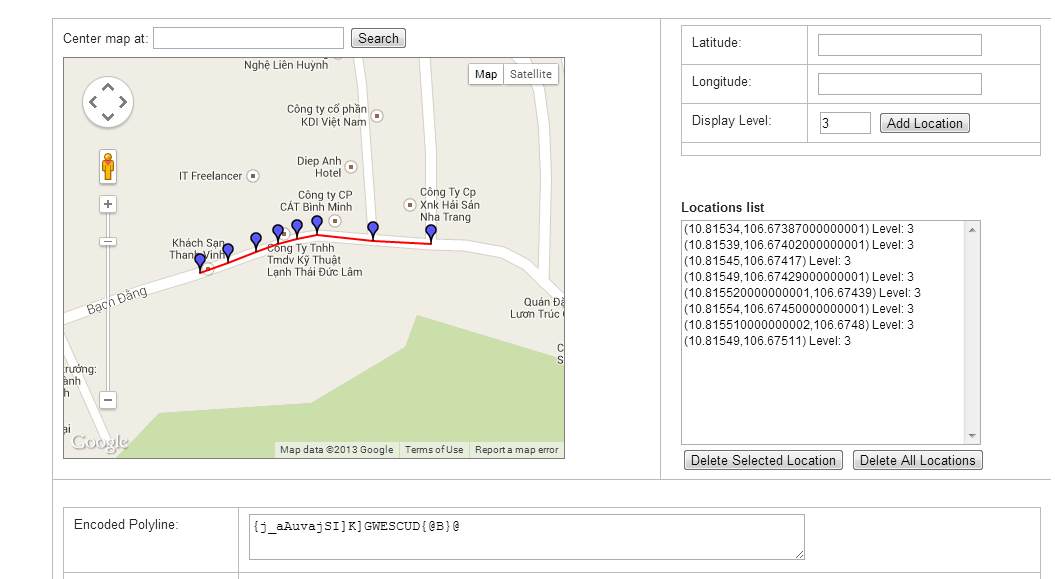
Vậy ta có giải thuật sau khi lấy được giá trị của points trong Object polyline như sau:



Ta tạo một List các đối tượng LatLng để tham chiếu kết quả trả về từ hàm decodePoly(String), là hàm giải mã mà ta đã nói đến ở trên. Sau đó, ta sẽ có một vòng lặp quét qua hết chiều dài của List này để thêm các giá trị Latitude & Longitude lấy được trong mỗi đối tượng LatLng vào biến HashMap, sau đó add các HashMap này vào List path, cuối cùng add List path vào List routes. Vậy hàm parse(JSONObject) của ta sẽ trả về kiểu giá trị List<List<HashMap<String, String>>>.

Ngoài ra, ta cũng viết một hàm có chức năng tương tự với tên là parseStartStopLocation(jObject), nhưng với kiểu trả về đơn giản hơn, chỉ là List<LatLng> để chứa tất cả các điểm trung gian. Ta sẽ dùng nó để tính toán các góc quay sau này. Và một điều khác biệt là ta sẽ bỏ qua những end\_location của các Steps, vì ta thấy end\_location của Steps này sẽ là start\_location của Steps sau, và sẽ gây ra lỗi khi có hai điểm trùng nhau, giải thuật điều khiển sẽ bị ảnh hưởng, và ta chỉ lấy end\_location của Steps cuối cùng. Ở hàm trên ta không quan tâm điều này vì khi vẽ lộ trình, hai điểm trùng nhau không gây ra lỗi.

Ta cùng xem hình ảnh thực tế sau khi giải mã, thông qua trang web giải mã trực tuyến của Google, khi ta cung cấp chuỗi mã hóa, Google sẽ xử lý giải mã và cho ta các tập hợp tọa độ cũng như hình ảnh trực quan bằng các Marker trên bản đồ:



Tiếp đến, với một List chứa các HashMap chứa các thông tin về Latitude & Longitude của các điểm trung gian, ta hoàn toàn có thể vẽ được lộ trình trên Google Map bằng các hàm đơn giản của API.

Ở Activity chính (MainActivity.java, là Activity của AutoMode), sau khi ta chạy AsyncTask để download nội dung file JSON, ta sẽ nhận được một đối tượng String chứa nội dung đó. Sau khi việc download hoàn tất, tức là ở trong hàm onPostExecute(Result), ta sẽ có tham số Result được truyền vào, ta sẽ tiếp tục dùng tham số này (chính là đối tượng String) để tạo một JSON Object và thực hiện parse dữ liệu theo thuật toán ở phía trên. Và sẽ tốt hơn nếu ta cũng thực hiện nó ở một AsyncTask khác để không làm ảnh hưởng hệ thống. Ta sẽ tạo một AsyncTask mới và đặt tên là ParserTask, dùng để gọi hàm parse(JSONObject) mà ta định nghĩa ở trên, sau đó xử lý kết quả và vẽ lộ trình lên bản đồ.



Ở hàm doInBackground, ta truyền tham số kiểu String, chính là đối tượng String chứa nội dung file JSON, và ta tạo một JSONObject từ đối tượng String đó, sau đó tạo một đối tượng DirectionsJSONParser, và gọi hàm parse(JSONObject) với tham số là JSONObject vừa tạo, và dùng một đối tượng List<LatLng> để lưu kết quả trả về của hàm parseStartStopLocation(jObject) mà ta sẽ dùng để tính góc khi điều khiển.

Ta đã có được kết quả trả về từ hàm doInBackground, ta sẽ lập trình hàm onPostExecute(Result) để xử lý kết quả này:

Làm ngược lại với các bước tạo nên ArrayList ở trên, ta sẽ thu được một tập hợp HashMap, từ các HashMap này, ta sẽ lấy được các giá trị Latitude & Longitude để lưu vào một List<Lat, Lng>.

Ta sẽ vẽ đường thẳng lên Google Maps thông qua đối tượng PolylineOptions, ta thêm tất cả các đối tượng địa điểm vào PolylineOptions thông qua hàm addAll(List<Lat,Lng>). Sau đó ta có thể tùy chỉnh thêm màu sắc cũng như bề rộng của nét vẽ:



Sau khi có được đối tượng PolylineOptions rồi, ta sẽ add vào Google Maps:



Như vậy ta đã khảo sát xong những tính năng cơ bản của Google Maps API trong việc:

* Hiển thị bản đồ lên giao diện của Activity.
* Thêm các Marker trên bản đồ.
* Download được file JSON chứa thông tin lộ trình giữa hai địa điểm.
* Viết các hàm parse để trích ra được thông tin của Latitude và Longitude của các điểm trên lộ trình đó.
* Vẽ các đoạn thẳng nối giữa các điểm, cho ta cái nhìn trực quan về lộ trình.

Ở phần tiếp theo, ta sẽ tiếp tục khảo sát module cuối cùng cần thiết lập, đó chính là đọc các giá trị cảm biến của thiết bị Android trong việc xác định góc phương vị hiện tại. Sau đó ta sẽ tiếp tục khảo sát giải thuật điều khiển bằng việc kết hợp các module này.

1. **Sử dụng các cảm biến của thiết bị Android để tính góc phương vị**

Module cuối cùng mà ta cần, chính là việc tính được góc quay của thiết bị. Có rất nhiều cách để làm được việc này, ta có thể dùng hàm getBearing() của đối tượng Location để tính toán, tuy nhiên, đáp ứng sẽ tùy theo sự cập nhật nhanh hay chậm của vị trí, và thường thời gian cập nhật vị trí không nhanh, dẫn đến kết quả góc đọc về chậm gây ảnh hưởng thuật toán điều khiển. Có một cách khác tốt hơn là ta sử dụng những cảm biến của thiết bị Android và có thể tính góc phương vị một cách rất dễ dàng mà lại cho ra kết quả gần như ngay lập tức.

Để có thể sử dụng và đọc kết quả của các cảm biến, ta cần một đối tượng SensorManager để lấy Service của hệ thống như sau:



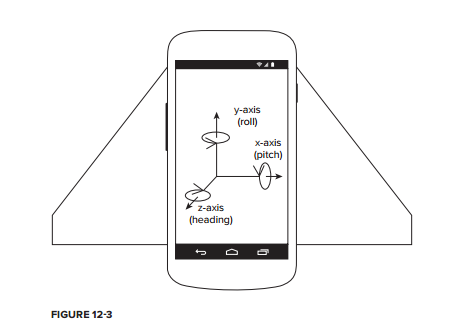
Sau đó ta có thể dùng đối tượng này để sử dụng các tính năng của cảm biến trong thiết bị, như tìm kiếm các cảm biến trong thiết bị, đọc kết quả của các cảm biến, …

Trước khi tính toán góc phương vị của thiết bị, ta sẽ tìm hiểu sự định hướng tự nhiên (natural orientation) của thiết bị trước. Ta sẽ dùng ba trục tọa độ để biểu diễn sự định hướng của thiết bị. Và phương định hướng tự nhiên sẽ là khi cả ba tọa độ này đều bằng 0. Đó là khi thiết bị nằm trên một mặt phẳng và hướng về hướng Bắc.

Để tính được sự định hướng của thiết bị, ta sẽ sử dụng giá trị đo được của cảm biến Magnetic Field (có chức năng như một compass điện tử) và cảm biến Accelerometer (dùng để tính giá trị Pitch và Roll).

Ta sẽ cùng khảo sát sự thay đổi định hướng trong không gian thông qua ba trục X Y Z tương ứng với Pitch, Roll và Azimuth.

* Pitch: là góc quay xung quanh trục X.
* Roll: là góc quay xung quanh trục Y.
* Azimuth: là góc quay xung quanh trục Z, trong phạm vi luận văn ta chỉ chú ý đến giá trị này là đủ. Trong đó 0 độ tương ứng với hướng Bắc, 90 độ tương ứng là hướng Đông, 180 độ tương ứng là hướng Nam, 270 độ tương ứng là hướng Tây.



* Tính toán góc phương vị dựa trên cảm biến Magnetic Field và Accelerometer:

Vì ta đang sử dụng hai cảm biến này, nên ta sẽ phải khai báo sử dụng chúng bằng đối tượng Sensor và lập trình một bộ Listener để xử lý khi giá trị các cảm biến này thay đổi.





Ta đã tạo hai đối tượng Sensor để tham chiếu đến hai cảm biến cẩn sử dụng, và đọc về giá trị của chúng khi giá trị thay đổi. Tiếp theo ta sẽ dùng giá trị đó để tính được ma trận quay của thiết bị thông qua hàm getRotationMatrix, với thông số là một Array 9 phần tử, giá trị của cảm biến Accelerometer và Magnet Field.



Ta có thể tính được các góc quay Pitch, Roll và Azimuth khi có được giá trị inR trả về. Tuy nhiên, đối với Android, khi sự định hướng của thiết bị và của màn hình hiển thị sẽ khác nhau, do khi thiết bị thay đổi thì nó vẫn sử dụng hệ tọa độ không đổi, vì vậy ta sẽ phải tính toán lại hệ tọa độ này, và quy đổi tương đối giữa inR nhận được ở trên thành một giá trị khác tương ứng với sự thay đổi đó.



Đầu tiên ta sẽ phải tính lại tọa độ X và Y, ví dụ ở trên là trong trường hợp thiết bị quay sang trái 90 độ, khi đó giá trị X sẽ thay đổi thành Y, và Y thay đổi thành –X. Tùy vào sự thay đổi định hướng của thiết bị mà ta phải tính toán lại cho phù hợp. Sau đó ta sẽ tính lại ma trận thông qua hàm remapCoordinateSystem, và lấy được kết quả của Pitch, Roll và Azimuth thông qua hàm getOrientation.



Tham số thứ hai của hàm getOrientation là một Array 3 phần tử thuộc kiểu float và đơn vị radian. Trong đó thứ tự tương ứng là góc Azimuth, Pitch và Roll. Ta sẽ quy đổi dễ dàng từ radian thành độ bằng hàm toDegree:



Như vậy ta đã hoàn thành xong việc khảo sát cách thức khai báo và sử dụng cảm biến Accelerometer và Magnetic Field để tính toán sự định hướng của thiết bị trong không gian. Và đến lúc này ta đã có đầy đủ công cụ để tiến hành các thuật toán điều khiển.

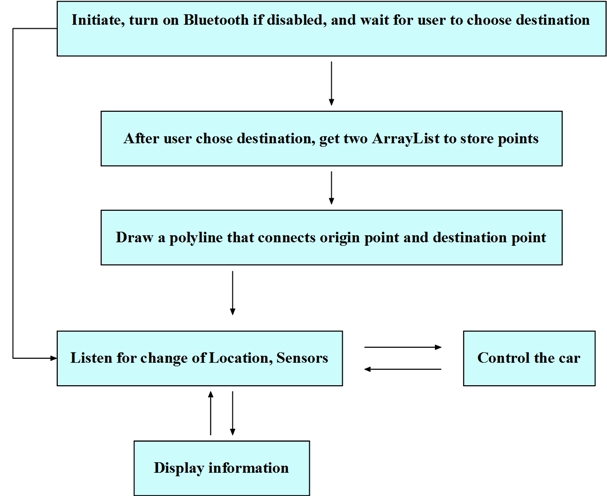
Phần tiếp theo ta sẽ đến với việc kết hợp các module LBS, Google Maps và các cảm biến để có thể đưa ra được thuật toán điều khiển phù hợp nhất.

1. **Thuật toán điều khiển ở chế độ Auto**

Ở các phần trước, ta đã khảo sát những module cần thiết dùng để sử dụng trong giải thuật điều khiển, bao gồm:

* Location Based Service: dùng để xác định vị trí hiện tại của thiết bị. Tọa độ của địa điểm hiện tại sẽ giúp ta nhận biến được vị trí của xe trong lộ trình, từ đó nhận biết được tín hiệu điều khiển như thế nào cho phù hợp.
* Google Maps API: dùng để vẽ bản đồ Google Map lên trên giao diện của Activity, giúp ta có cái nhìn trực quan hơn về các địa điểm. Đây cũng là giao diện mà ta sẽ chọn điểm đến thông qua việc tạo Marker trên bản đồ, sau đó ta sẽ nhận được một ArrayList tham chiếu đến các tọa độ của các điểm trung gian. Từ đó vẽ được hình ảnh trực quan trên bản đồ, đó là các đường kỷ hà nối từ điểm đầu đến điểm đích, đi qua các điểm trung gian. Và ta còn sử dụng ArrayList đó để tính toán góc mà xe sẽ phải quay khi di chuyển.
* Cảm biến Magnet Field và Accelerometer của thiết bị Android. Đây là hai cảm biến sẽ giúp chúng ta tìm được góc phương vị của thiết bị (tương ứng là góc so với gốc là phương Bắc). Ưu điểm của việc dùng hai cảm biến của thiết bị là thời gian đọc tín hiệu rất nhanh, giúp thuật toán điều khiển chính xác hơn, so với việc tính góc phương vị bằng hàm của giải thuật LBS.

Ta sẽ khảo sát sâu hơn về sự kết hợp giữa các module trên qua sơ đồ sau:



Đầu tiên, khi khởi tạo chương trình, hệ thống sẽ yêu cầu bật Bluetooth nếu như Bluetooth chưa được bật, sau đó, đồng thời hệ thống bắt đầu cập nhật vị trí thông qua Location Based Service và cập nhật các cảm biến. Hệ thống cũng sẽ đợi người dùng chọn điểm đến trên bản đồ. Ta có thể chọn điểm đến bằng cách chạm vào bản đồ để tạo một Marker tại nơi ta cần đến. Vì vậy, ta sẽ phải khai báo một bộ onMapClickListener, sử dụng hàm onMapClick(LatLng) với tham số là đối tượng LatLng chứa thông tin của điểm đó trên bản đồ. Từ thông tin tọa độ ta sẽ vẽ được một Marker tương ứng, đồng thời truyền đối số cho hàm getDirectionsUrl. Tọa độ điểm gốc chính là địa điểm hiện tại cung cấp bởi LBS, và địa điểm đến chính là địa điểm ta vừa chọn.

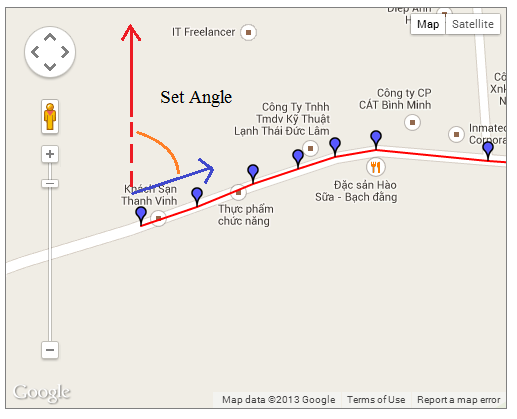
Sau đó thông qua các AsyncTask đã giới ở trên, ta có được hai ArrayList để chứa thông tin về các địa điểm trung gian. Nhờ đó ta sẽ vẽ được một polyline nối giữa hai địa điểm, và tính toán được giải thuật điều khiển Control Car.

Ngoài ra ta còn có một khối Display, dùng để quan sát các thông tin sau:

* Latitude và Longitude của địa điểm hiện tại.
* Góc quay sắp tới.
* Góc phương vị hiện tại.
* Khoảng cách giữa địa điểm hiện tại và địa điểm trung gian gần nhất.
* Tín hiệu điều khiển.

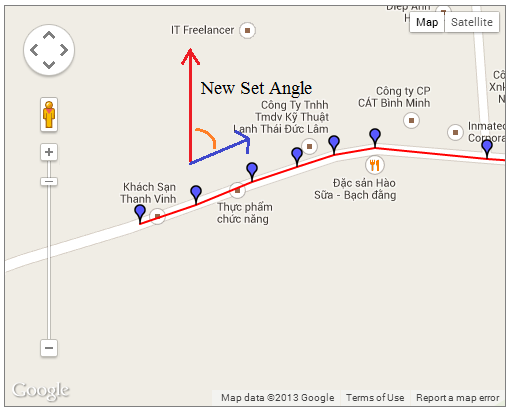
Trong phạm vi luận văn, những giá trị này chủ yếu để theo dõi sự hoạt động của ứng dụng, sẽ giúp ta trong việc điều chỉnh sai sót trong giải thuật.

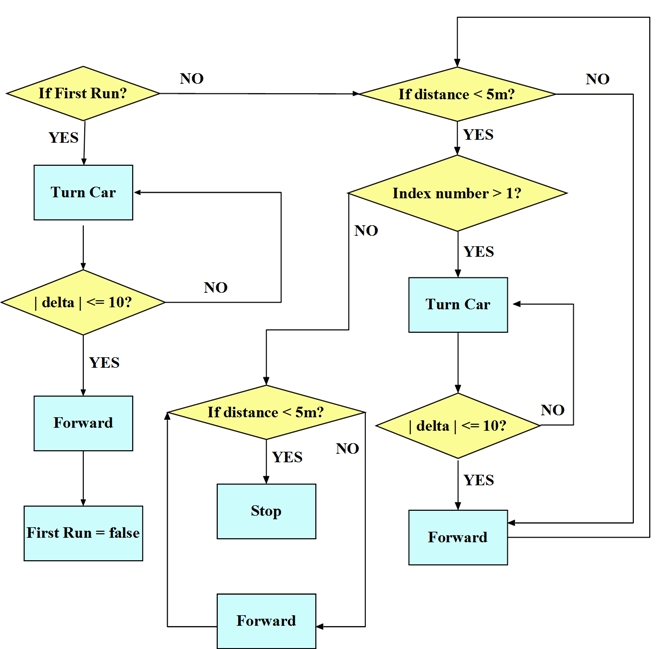
Tiếp đến ta sẽ phân tích khối Control Car:

Trước khi phân tích giải thuật của khối này, ta cần một ví dụ trực quan về giải thuật điều khiển phù hợp.

Như đã phân tích, sau khi ta chọn điểm đến, ta sẽ có một ArrayList chứa thông tin của tất cả các địa điểm ta cần đi qua, bao gồm cả điểm đầu và cuối. Giải thuật cơ bản của ta là, ta coi như giữa hai địa điểm liền kề nhau, xe sẽ di chuyển trên một đường thẳng. Khi xe đi đến cuối đoạn thẳng này, nó sẽ gặp đoạn thẳng nối giữa hai điểm tiếp theo. Và góc quay sẽ thay đổi, nhiệm vụ của ta là phải tính toán được góc quay này (Set Angle) và điều khiển xe khi nó đến các địa điểm nút (là nơi mà phương đường thẳng thay đổi).

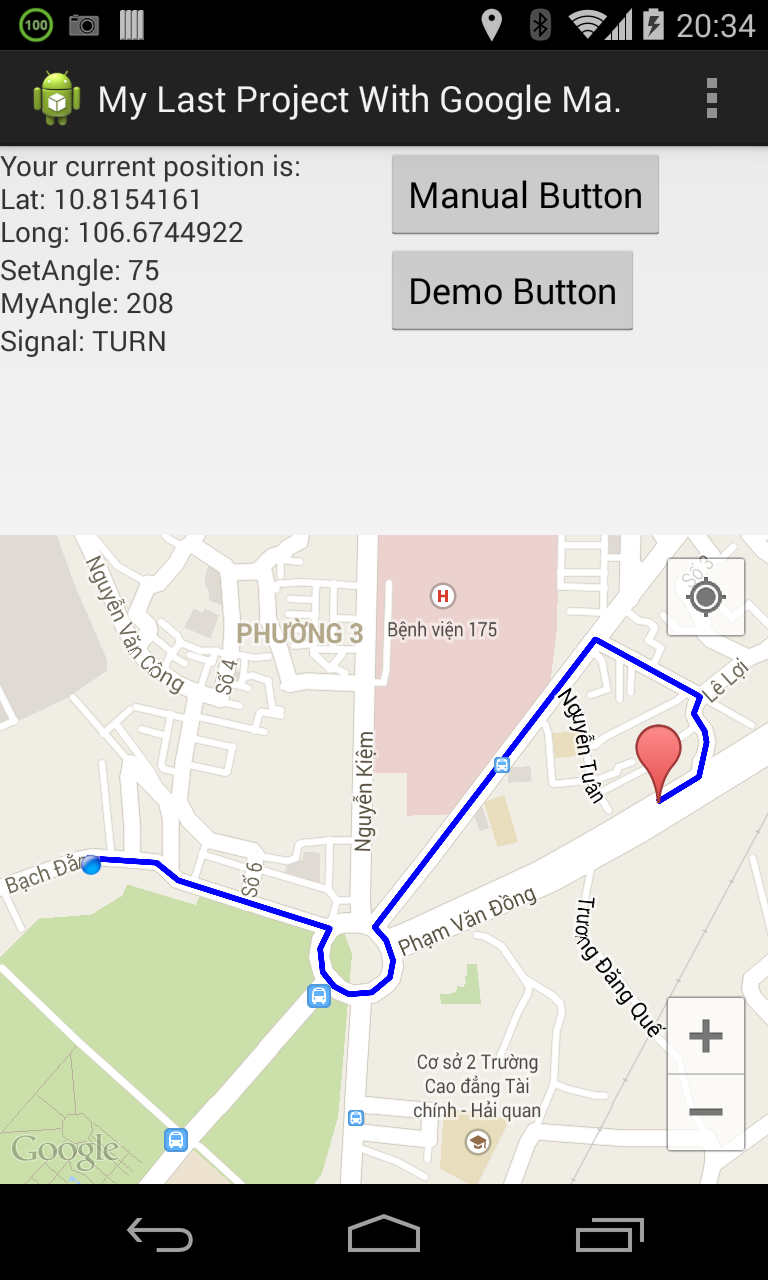
Ta sẽ có hai trường hợp:

* Khi xe khởi động: đây là trạng thái First Run (giá trị First Run là “true”), khi đó xe đang ở trạng thái dừng, và góc quay Set Angle sẽ là góc quay giữa địa điểm có index là 1 với địa điểm có index là 0 (chính là vị trí hiện tại). Ta cần tính toán góc này, và cho xe di chuyển xoay cho đến khi đạt đến góc quay đó. Sau đó, ta sẽ xuất tín hiệu cho xe đi thẳng, và cho trạng thái First Run thành “false”. Và ta cần xóa địa điểm hiện tại ra khỏi ArrayList, như vậy trở về sau, địa điểm có index là 0 sẽ là địa điểm tiếp theo trên lộ trình.
* Khi xe đang ở một vị trí bất kì trên lộ trình: dễ thấy là trạng thái này, xe sẽ đang đi thẳng. Ta cũng cần tính góc quay Set Angle giữa địa điểm index là 1 với địa điểm index là 0. Khi khoảng cách giữa vị trí hiện tại của xe với địa điểm index là 0 nhỏ hơn 5m, ta sẽ cho xe di chuyển xoay, đến khi đạt được góc quay này, và rồi tiếp tục đi thẳng, và xóa địa điểm có index là 0 ra khỏi Array.
* Khi Array chỉ còn một phần tử: khi này, ta đã đi qua hết tất cả các lộ trình, chỉ còn một điểm cuối, ta sẽ không cần tính góc quay nữa, và khi xe đi đến điểm cuối này, ta sẽ xuất tín hiệu dừng, và thông báo cho người dùng về việc kết thúc.



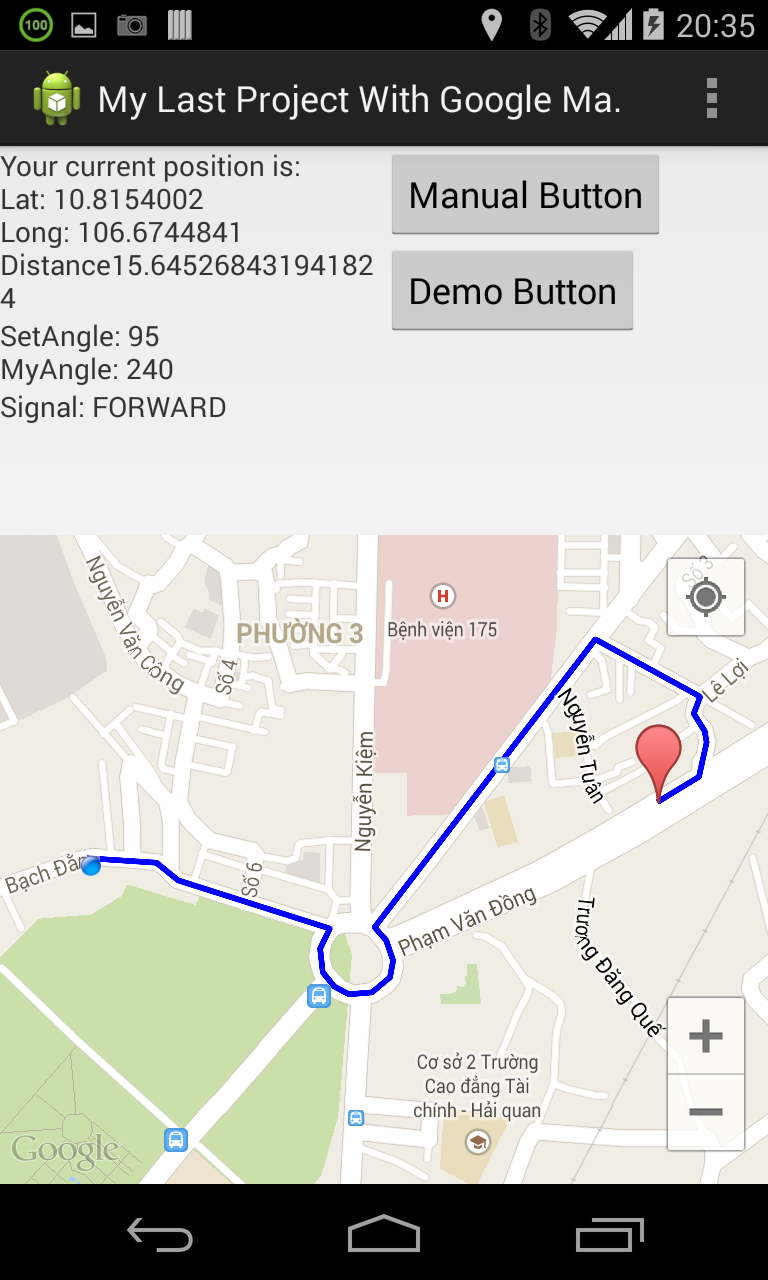
Lưu đồ giải thuật có thể mô tả như bên dưới:

Trong đó, delta chính là hiệu giữa góc phương vị và góc quay cần thiết kế tiếp. Ta cần hiệu của chúng nằm trong khoảng -10 độ đến 10 độ là đủ. Distance chính là khoảng cách giữa địa điểm hiện tại với địa điểm gần nhất với nó (chính là địa điểm có index 0 trong ArrayList).

Giao diện của MainActivity (AutoMode) khi ứng dụng hoạt động, bằng việc kết hợp các kiểu Layout mà ta có được giao diện như bên dưới:

Khi xe mới khởi động, tức là First Run = true:

Khi xe không còn ở First Run nữa, lúc này ta có thêm giá trị Distance là khoảng cách từ địa điểm hiện tại đến địa điểm trung gian gần nhất:



Nhận xét về phần ứng dụng phần mềm trên nền tảng Android:

Sau quá trình phát triển, ứng dụng về cơ bản đã hoàn chỉnh, đáp ứng tốt các yêu cầu của luận văn trong nhiệm vụ xuất tín hiệu điều khiển xe trong chế độ thủ công (Manual Mode) cũng như trong chế độ điều khiển tự động (Auto Mode). Tín hiệu xuất chính xác, đáp ứng tốt, giải thuật điều khiển cũng hoạt động trơn tru, đúng yêu cầu.

**Phần III. Phần cứng xe tự hành và vi điều khiển STM32F4**

Ứng dụng của luận văn là thiết kế một hệ thống xe tự hành có thể di chuyển trên đường thông qua tín hiệu điều khiển từ vi điều khiển STM32F4. Ngoài việc xây dựng phần mềm và giải thuật điều khiển, công đoạn thiết kế phần cứng cũng rất quan trọng, vì nó quyết định đến việc xe có vận hành đúng như tín hiệu đã điều khiển hay không.

Với yêu cầu như trên, ta sẽ xây dựng và thiết kế phần cứng như sau:

1. **Phần cơ khí**

Ứng dụng của ta là một xe mô hình, mô phỏng thực tế hoạt động của một ô tô bốn bánh, vì vậy ta cần thiết kế cơ khí sao cho xe có thể vận hành linh hoạt theo bốn tín hiệu điều khiển chính: Forward, Backward, Turn Left và Turn Right. Ta sẽ xây dựng phần cơ khí gồm:

* Một tấm Mica hình chữ nhật kích thước 40cm x25cm dùng để làm thân xe.
* Hai bánh xe omni, đây là bánh xe chuyển động đa hướng. Do ta chưa thể thực hiện hệ thống truyền động phức tạp như xe ô tô thông thường, nên để xe có thể rẽ trái và rẽ phải được, ta cần sử dụng hai bánh xe Omni ở phía trước, hai bánh xe này sẽ giúp điều hướng cho xe.
* Hai bánh xe nhôm đường kính 100mm có đai giảm tốc. Đây sẽ là hai bánh xe truyền động trực tiếp từ động cơ, và làm cho xe di chuyển theo hai hướng là tiến và lùi.
* Các hệ thống truyền động đơn giản để cố định và truyền động cho bánh xe.

Như vậy ta đã khảo sát xong những yêu cầu của phần cơ khí mà ta cần phải thiết kế và xây dựng. Tiếp theo, ta sẽ đến với phần thiết kế mạch điện cho xe.

1. **Phần mạch điện**

Cũng như phần cơ khí, phần mạch điện cũng góp vai trò vô cùng quan trọng trong việc thể hiện thuật toán điều khiển của chúng ta. Nếu mạch điện truyền tải tín hiệu không đúng thì sẽ dẫn đến đáp ứng của phần cơ khí sẽ không chính xác.

Phần cơ khí ta thiết kế hai bánh xe để truyền động, và ta sẽ sử dụng hai động cơ để truyền động cho các bánh xe. Động cơ ta chọn sử dụng là động cơ DC 24V, có gắn Encoder để hồi tiếp tín hiệu về.

Ta cần một mạch nguồn để cung cấp điện áp cho cả hệ thống, và phần mạch của chúng ta cần nhiều mức điện áp khác nhau: 24V cho driver, 5V cho mạch đệm và vi điều khiển, vì vậy ta cần thiết kế một mạch nguồn chỉnh lưu và có thể đưa ra nhiều mức điện áp, từ 24V đến 5V.

Do ta muốn điều khiển động cơ với nhiều tốc độ khác nhau và nhiều chế độ khác nhau, ta sẽ không cấp thẳng trực tiếp nguồn vào động cơ, mà sẽ cấp xung PWM từ vi điều khiển cho động cơ. Do dòng xuất ra từ vi điều khiển có dòng yếu và điện áp thấp, ta sẽ cần một mạch driver, vừa để giải quyết vấn đề điện áp (do vi điều khiển chỉ cung cấp tín hiệu điều khiển, còn điện áp 24V sẽ do mạch driver cấp cho động cơ), vừa giảm sức nặng cho giải thuật điều khiển động cơ.

Mạch driver ta sẽ dùng là mạch H-Bridge sử dụng IC LMD18200.

1. **Giới thiệu vi điều khiển STM32F4 và chương trình điều khiển**

* **Giới thiệu vi điều khiển STM32F4**

Vi xử lý STM32F4 (đầy đủ là STM32F407VG) dựa trên nhân ARM Cortex M4 có thể hoạt động ở tần số tối đa là 168MHz. Chip ARM Cortex M4 có tính năng FPU (Floating Point Unit) hỗ trợ tất cả các lệnh xử lý dữ liệu và các loại dữ liệu.

Họ vi xử lý STM32F4 bao gồm bộ nhớ tốc độ cao (1Mbyte Flash Memory, 192Kbyte SRAM), nhiều I/O và ngoại vi mở rộng được kết nối bởi hai bus APB, ba bus AHB và ma trận các bus AHB 32 bit.

Tất cả các vi xử lý thuộc họ STM32F4 đều có ba bộ ADC 12 bit, hai bộ DAC, bộ RTC tiết kiệm năng lượng, mười hai bộ timer 16 bit bao gồm hai timer PWM để điều khiển động cơ và hai timer 32 bit. Ngoài ra còn có:

* Ba bộ I2C.
* Ba bộ SPI, hai bộ I2S.
* Bốn bộ USART với hai bộ UART.
* Một bộ USB OTG.
* Hai bộ CAN.
* Một giao diện SDIO/MMC.
* Giao diện Ethernet và camera.

Họ vi điều khiển STM32F4 làm việc từ ở khoảng nhiệt độ từ -45 độ đến 105 độ, với nguồn cung cấp từ 1.8V đến 3.6V.

Với những tính năng mà STM32F4 cung cấp, ta có thể sử dụng họ vi điều khiển này trong các ứng dụng như:

* Điều khiển động cơ.
* Thiết bị y khoa.
* PLC, Inverter, CB.
* Máy in, máy Scan.
* Hệ thống báo động.
* Thiết bị audio gia đình.

Và trong phạm vi luận văn, ta sẽ sử dụng vi điều khiển STM32F407VG để điều khiển hoạt của động cơ bằng cách xuất xung PWM.

* **Các tính năng của STM32F4 cần sử dụng để hoàn thành giải thuật điều khiển.**

Để điều khiển được động cơ, hay nói đúng hơn là xuất tín hiệu điều khiển cho driver, ta cần các tín hiệu điều khiển là PWM (xung điều rộng), DIR (một bit tín hiệu điều khiển hướng), và BREAK (High tương ứng với việc dừng động cơ).

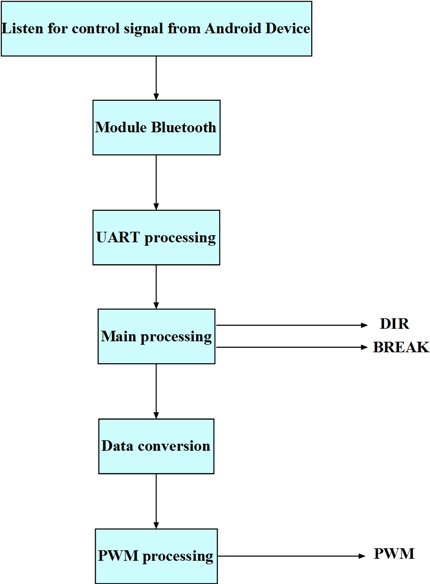
Ta có hai động cơ cần điều khiển, do vậy ta cần hai driver để điều khiển chúng. Vì vậy ta sẽ có sáu chân I/O của vi điều khiển dùng để xuất tín hiệu cho driver.

Trong số sáu tín hiệu đó, hai tín hiệu PWM là hai tín hiệu quan trọng nhất, và ta sẽ sử dụng hai bộ Compare của một Timer để hoàn thành công việc này.

Với bốn tín hiệu còn lại, chỉ thuẩn túy là tín hiệu I/O đơn giản, và ta có thể dễ dàng thiết lập.

Thêm vào đó, vì vi điều khiển sẽ nhận tín hiệu điều khiển từ thiết bị Android, thông qua giao tiếp UART, nên ta cũng cần thiết lập cấu hình UART của vi điều khiển, viết các hàm để nhận tín hiệu, chuyển đổi tín hiệu điều khiển sao cho thích hợp và chính xác.

**Ta sẽ có lưu đồ giải thuật như bên dưới:**



Như đã giới thiệu ở mục phần cứng phía trên, do mục đích điều khiển không dây, ta sẽ dùng module Bluetooth HC-05 để nhận dữ liệu từ thiết bị Android và chuyển đổi thành tín hiệu UART truyền đến vi điều khiển.

Sau khi nhận tín hiệu UART từ module Bluetooth, vi điều khiển sẽ đọc dữ liệu về, sau đó dữ liệu sẽ được xử lý trong chương trình chính để xuất được giá trị DIR và BREAK, phần dữ liệu còn lại (như đã đề cập trong phần lập trình Android) sẽ được chuyển đổi thành kiểu dữ liệu số (từ ASCII thành Integer) và thông qua thiết lập PWM ta sẽ xuất được tín hiệu điều khiển PWM.

Tiếp đến ta sẽ đi chi tiết vào từng phần trong giải thuật điều khiển.

* **Thiết lập GPIO:**

GPIO – General Purpose I/O, họ vi điều khiển STM32F4 bao gồm các GPIO 16 bit, được thiết lập thông qua bốn thanh ghi thiết lập 32 bit: GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR và GPIOx\_PUPDR, hai thanh ghi dữ liệu 32 bit: GPIOx\_IDR và GPIOx\_ODR, một thanh ghi set/reset dữ liệu 32 bit GPIOx\_BSRR, một thanh ghi khóa 32 bit: GPIOx\_LCKR, và hai thanh ghi chọn chức năng thay thế 32 bit GPIOx\_AFRL và GPIOx\_AFRH.

Khi thiết lập cho GPIO, ta cần biết các thông số và thanh ghi như sau:

* + Trạng thái output: push-pull hoặc open drain + pull-up/down.
  + Lựa chọn tốc độ cho các I/O.
  + Thanh ghi GPIOx\_ODR là thanh ghi chứa dữ liệu output của I/O.
  + Thanh ghi GPIOx\_IDR là thanh ghi chứa dữ liệu input của I/O.
  + Set hoặc reset bit của output dựa vào thanh ghi GPIOx\_BSRR trong đó 16 bit thấp (BSRRL) là set và 16 bit cao (BSRRH) là reset.

Ví dụ khi thiết lập chân PB11 dùng làm output, tốc độ 100MHz:



Trong phạm vi luận văn, ta sẽ sử dụng các chân GPIO tương ứng với các tín hiệu điều khiển như sau:

* PB2: DIR của động cơ trái.
* PB10: DIR của động cơ phải.
* PB9: BREAK của động cơ trái.
* PB11: BREAK của động cơ phải.
* **Thiết lập UART:**

Bất cứ giao tiếp UART nào cũng cần hai chân quan trọng, đó là chân nhận dữ liệu – Receive Data In (RX), và chân truyền dữ liệu – Transmit Data Out (TX).

Thông qua các chân này, dữ liệu được truyền và nhận theo các frame như sau:

* Một quãng Idle trước khi truyền và nhận.
* Một bit bắt đầu (Start bit).
* Dữ liệu (8 hoặc 9 bit) trong đó LSB được truyền trước.
* 0.5, 1, 1.5 hoặc 2 bit dừng báo hiệu xong một frame.
* Thanh ghi trạng thái (USART\_SR).
* Thanh ghi dữ liệu (USART\_DR).
* Thanh ghi tốc độ baud (USART\_BRR).

Ở trên là các thông số thường phải được khai báo khi muốn thiếp lập truyền nhận dữ liệu thông qua UART, không chỉ với vi điều khiển STM32F4.

Trong phạm vi luận văn, ta sẽ sử dụng USART1, đối với USART 1 thì chân RX sẽ là chân PB7 và TX sẽ là chân PB6, vì vậy ta cần làm hai công việc: thứ nhất là thiết lập cấu hình cho USART 1, thứ hai là thiết lập GPIO cho chân PB6 và PB7 hoạt động ở chế độ chức năng thay thế (Alternative Function).

Ta sẽ thiết lập PB6 và PB7 ở chế độ output và Alternative Function như sau:



Ta thiết lập USART 1 với tốc độ baud 9600, độ dài dữ liệu là 8 bit, 1 bit dừng, không sử dụng bit parity và không sử dụng flow control của phần cứng, và cho phép ngắt của USART 1 như sau:



Tiếp đến ta sẽ xây dựng hàm ngắt của USART 1 để xử lý dữ liệu nhận được từ thiết bị Android. Vì thiết bị Android ta sẽ lập trình để gửi 10 ký tự có cấu trúc là L+(-)xxxR+(-)xxx, nên ta cần một Array 10 ký tự ở chương trình của vi xử lý để lưu 10 ký tự này.

Việc viết chương trình ngắt tương đối đơn giản, ta tạo một Array received\_string[10] gồm 10 phần tử, và dùng một biến đếm từ 0 đến 9 để lần lượt lưu các byte vào từng phần tử.



Việc lưu 10 ký tự vào một biến sẽ giúp ta có thể lập trình dễ dàng hơn cho giải thuật điều khiển, vì ta chỉ cần rút các phần tử tại index mà t muốn để xử lý.

* **Thiết lập Timer và PWM:**

Như đã giới thiệu ở trên, ta sẽ thiết lập ngõ ra PWM dựa vào thiết lập Timer và các bộ Compare của Timer.

Trong phạm vi luận văn ta sẽ sử dụng Timer 4 để thiết lập ngõ ra PWM để điều khiển động cơ. Timer 4 là một timer 16 bit, với giá trị prescaler có thể lập trình được từ 1 đến 65536. Timer 4 có bốn kênh cho các mode:

* Input Capture.
* Output Compare.
* PWM Generation.
* One pulse mode.

Ta sẽ sử dụng 2 kênh Output Compare tương ứng là 2 chân PWM dùng để xuất xung PWM đến driver. Trước khi thiết lập Output Compare, ta cần thiết lập Timer. Trong đó các thông số cần quan tâm là Prescaler, Clock Division, Counter Mode và Timer Period.

Prescaler là giá trị mà xung nhịp của nguồn bị chia ra (từ 1 đến 65536), Clock Division là số chia xung đếm của Counter, Counter Mode ta chọn là đếm lên, và chu kì của Timer ta đặt là 256 chu kỳ (chu kỳ sau khi đã chia xung nhịp nguồn cho Prescaler).

Ta thiết lập Timer 4 như sau:



Sau khi đã thiết lập cơ bản cho Timer 4, ta sẽ thiết lập Ouput Compare Mode. Chức năng này dùng để điều khiển dạng sóng ngõ ra. Khi giá trị của Counter bằng với giá trị đặt sẵn để so sánh trong thanh ghi Compare, và tùy thuộc vào cách thức tác động của hệ thống lên các bit OCxM, CCxP mà ngõ ra sẽ có hình thức thay đổi tương ứng. Ở đây ta lựa chọn OCMode là PWM, tức là khi giá trị Counter bằng giá trị thanh ghi Compare, nó sẽ xuống mức 0, còn lại là mức 1:



Như vậy, ta đã hoàn thành xong phần giới thiệu các bước thiết lập các GPIO, USART, Timer và PWM cần thiết để thực hiện giải thuật. Tiếp theo ta sẽ xét tiếp thuật toán giải thuật để có được đáp ứng cuối cùng đến từ phần cứng.

* **Thuật toán điều khiển phần cứng**

Sau khi đã thiết lập xong các công đoạn cần thiết cho vi điều khiển, ta sẽ tiến hành viết giải thuật điều khiển. Để giải thuật được gọn và dễ chỉnh sửa, ta sẽ viết thêm các hàm nhỏ để điều khiển bốn thao tác Forward, Backward, Turn Left và Turn Right. Ta sẽ chỉ quan tâm đến tín hiệu DIR và BREAK, tín hiệu PWM ta có thể xuất liên tục.

Ta sẽ cần một hàm nhỏ để chuyển đổi tín hiệu PWM gửi từ thiết bị Android từ ký tự ASCII thành một số Integer tương ứng.

Ví dụ cho giá trị PWM của bánh trái:



Ta đã có đủ công cụ cần thiết để có thể đi đến thuật toán điều khiển như sau:

