RamaniLocationManager

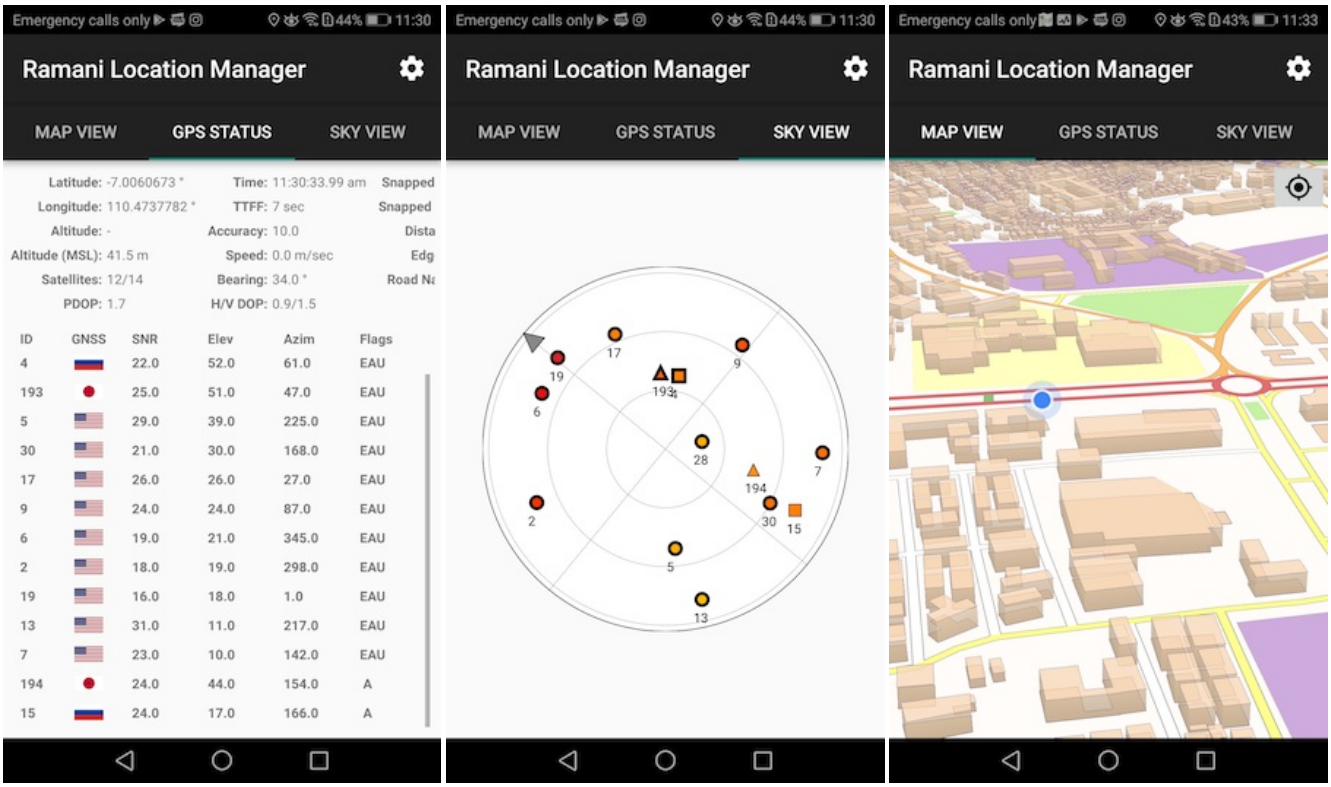
RamaniLocationManager provides a custom Android framework to obtain geo-location information supporting Unassisted GPS and Assisted GPS conditions, incl. the ability to simulate positioning uncertainty in these use cases.

Contents

Product summary

- **Product description**. RamaniLocationManager is a library to make it easy for Android developers to build and test location-aware applications, e.g. enabling features to work even when GPS-reception is problematic.
- **Value proposition**. Build and test the location-aware App features safely and easily using RamaniLocationManager.
- **Company description**. RamaniLocationManager provides a better framework to obtain the geo-location information on Android devices, supporting Unassisted GPS and Assisted GPS conditions, incl. the ability to simulate uncertainty in use cases.
- **Mission**. RamaniLocationManager makes it easy for the Android App developers to build robust location-aware features.

Screen Captures



Sample App

- Download the Application on any supported device and see how to use it on Getting Started using the following Google Play Store link:

Core Functionality

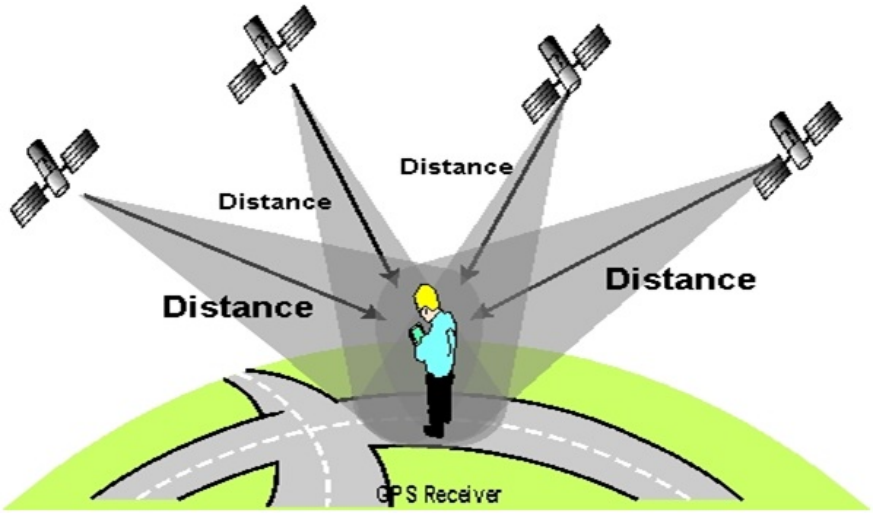
- Read GPS-status and be able to simulate various adverse GPS-reception condition on the GPS-status
- Provide inertia sensor-fusion to estimate geo-location under GPS-unassisted navigation use cases
- Provide Snap-to-road estimated geo-location under GPS-unassisted navigation use cases
- Read Satellites/GNSS status (global navigation satellite system)
- Setting Up MockLocation
- Snap-To-Road

Background Knowledge

What is GPS?

GPS or Global Positioning System is a satellite navigation system that furnishes location and time information in all climate conditions to the user. GPS is used for navigation in planes, ships, cars and trucks also. The system gives critical abilities to military and civilian users around the globe. GPS provides continuous real time, 3-dimensional positioning, navigation and timing worldwide…

How GPS Determines a Position:



What is GNSS?

Global Navigation Satellite System (GNSS) refers to a constellation of satellites providing signals from space that transmit positioning and timing data to GNSS receivers. The receivers then use this data to determine location.

By definition, GNSS provides global coverage. Examples of GNSS include Europe's Galileo, the USA's NAVSTAR Global Positioning System (GPS), Russia's Global'naya Navigatsionnaya Sputnikovaya Sistema (GLONASS) and China's BeiDou Navigation Satellite System.

The performance of GNSS is assessed using four criteria:

1. **Accuracy**: the difference between a receiver's measured and real position, speed or time;
2. **Integrity**: a system's capacity to provide a threshold of confidence and, in the event of an anomaly in the positioning data, an alarm;
3. **Continuity**: a system's ability to function without interruption;
4. **Availability**: the percentage of time a signal fulfils the above accuracy, integrity and continuity criteria.

System	GPS	GLONASS	BeiDou	Galileo	NAVIC
Owner	United States	Russian Federation	China	European Union	India
Orbital altitude	20, 180 km (12, 540 mi)	19, 130 km, (11, 890 mi)	21, 150 km, (13, 140 mi)	23,222 km , (14, 429 mi)	36, 000 km, (22,000mi)
Period	11.97 h, (11 h 58 min)	11.26 h, (11 h 16 min)	12.63 h, (12 h 38 min)	14.08 h, (14h 5min)	
Number of satellites	32 (at least 24 by design)	28 (at least 24 by design) including: 24 operational 2 under check by the satellite prime contractor 2 in flight tests phase	5 geostationary orbit (GEO) satellites, 30 medium Earth orbit (MEO) satellites	4 in-orbit validation satellites + 8 full operation capable satellites in orbit 22 operational satellites budgeted	Total: 7 In Orbit: 7
Frequency	1.57542 GHz (L1 signal) 1.2276 GHz (L2 signal)	Around 1.602 GHz (SP) Around 1.246 GHz (SP)	1.561098 GHz (B1) 1.589742GHz (B1-2) 1.20714 GHz (B2) 1.26852 GHz (B3)	1.164-1.215 GHz (E5a and E5b) 1.260-1.300 GHz (E6) 1.559-1.592 GHz (E2-L1-E11)	L5-band 1164.45-1188.45 MHz S-band 2483.5-2500 MHz
Status	Operational	Operational	22 satellites operational, 40 additional satellites 2016-2010	8 satellites operational, 22 additional satellites 2016-2020	Operational

System Requirements

Supported Sample Android version:

- The minimum Android OS to install RamaniLocationManager-Sample is Jelly Bean (4.1 and up).

Supported Tools:

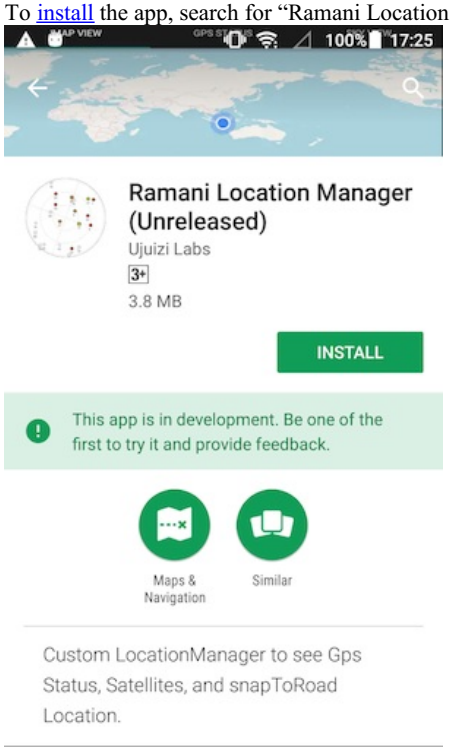
- Minimum requirement to build with project is Android Studio 3.0 and up

Software Dependencies

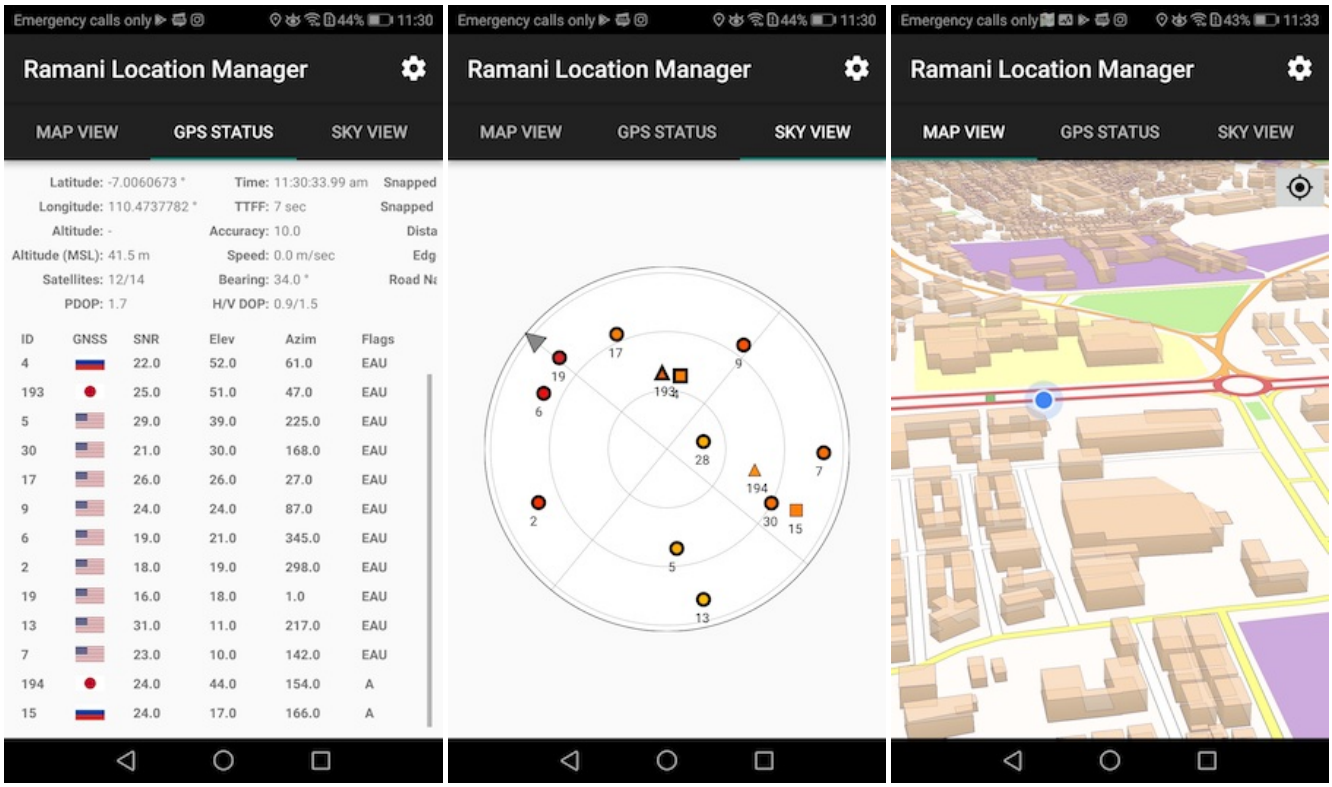
- RAMANI Maps-API
- RAMANI Feedback
- Gitlab API
- Graphhopper (Core, Map-Matching)

Getting Started

- To install the app, search for “Ramani Location Manager” (without the quotes) in the Google Play Store or click here and click on INSTALL:



- Open the App, and from the Main View, you can switch between 3 tabs to see the MapView, Gps status, and Sky view.



Using the App

Ramani Location manager library is can be see on Archiva (see this url for mor details
https://team.ujuizi.com/archiva/repository/internal/com/ujuizi/ramani/location-manager/)

Set SDK Version

This is our compileSDKVersion, minSdkVersion, and targetSdkVersion

```
android {
    compileSdkVersion 27

    defaultConfig {
          minSdkVersion 16
          targetSdkVersion 27
        ...
        ...
    }
}
```

Add it on Project using Android Studio

To add this project as artifact depedency on your project :

Add this line on your project, build.gradle :

```
allprojects {
    repositories {
        jcenter()
        maven {
            url 'http://team.ujuizi.com/archiva/repository/internal'
        }
    }
}
```

Then on your dependency app build.gradle
add this line :

```
dependency{
    ....
    implementation 'com.ujuizi.ramani:location-manager:0.0.3-alpha'
    ....
}
```

Sample Code

See the example code section below.

Reading Satellites/GNNS Status Data

```
//set up GPSStatusData
       GPSStatusData gpsStatusData = new GPSStatusData(mActivity);
           //call this to handle GPS Satellite Data Listener
           gpsStatusData.addGpsStatusListener(new GPSStatusChangedListener() {
                   @Override
                   public void onGPSStatusEvent(int event, String status) {
                        Log.e("Satellites","Gps Status Event : "+status);
                   }
                   @Override
                   public void onGetSatelliteTypes(SatelliteType[] satelliteTypes,
                   int totalUsedInFixCount, int totalSv) {
                         // read total satellites
                         Log.e("Satellites","Total Satellite : "+totalSv);
                         // read total satellites used in Fix
                         Log.e("Satellites","Total Satellite : "+totalUsedInFixCount);
                         // read satellites
                         for (SatelliteType satelliteType: satelliteTypes){
                             Log.e("Satellites",satelliteType.toString());
                         }
                   }
                   @Override
                   public void onNmeaMessage(String message, long timestamp) {
                          //Get Altitude Mean Sea Level
                          if (message.startsWith("$GPGGA") || message.startsWith("$GNGNS")) {
                              Double altitudeMsl = getAltitudeMeanSeaLevel(message);
                          if (altitudeMsl != null) {
                              Log.e("Satellites","AltitudeMSL : "+String.valueOf(altitudeMsl));
                              }
                          }
                          //Get Dilution of Precision (DOP), Horizontal DOP, and Vertical DOP
                         if (message.startsWith("$GNGSA") || message.startsWith("$GPGSA")) {
                              DilutionOfPrecision dop = getDop(message);
                              if (dop != null) {
                                  Log.e("Satellites","DOP : "+String.valueOf(dop.getPositionDop()));
                                  Log.e("Satellites","Horizontal and Vertical DOP :
                                  "+String.valueOf(dop.getHorizontalDop())+" , "+
                                  String.valueOf(dop.getVerticalDop()));
                               }
                          }
                   }
               });
               //add this if you want to get NMea Message
               gpsStatusData.addNmeaStatusListener();
```

Setup Routing Data

```
       //Use this if you want to set in UI Thread
            OfflineRouting offlineRouting = new OfflineRouting("Your Path",
            new OfflineRouting.OfflineRoutingListener() {
                      @Override
                      public void onRoutingDone(boolean isReady) {
                          Log.e ("OfflineRouting","Routing ready : "+isReady);
                      }
               });
            //use this if you want to set in background thread
            OfflineRouting offlineRouting = new OfflineRouting("Your Path");
            boolean status = offlineRouting.checkGraphStorage();
```

Snap-to-road

```
       // use this if you want to use in UI Thread
       SnapToRoad snapToRoad = new SnapToRoad(offlineRouting);
       snapToRoad.getSnapToRoadInBackground(location, new SnapToRoadListener(){
           @Override
           public void onSuccesSnapped(SnapToRoadModel snappedLocation) {
          Log.e("SnapToRoad", "onSuccess : " + snappedLocation.toString());
           }
           @Override
           public void onFailedSnapped(String failedMessage) {
               Log.e("SnapToRoad", "onFailedSnapped : " + failedMessage);
           }
       });
       //call this if you want to call in Backgroundtask
       SnapToRoadModel snaptoroadModel = snaptoroad.getSnapToRoad(location);
```

Mock Location

```
      MockLocation mockLocation = new MockLocation(this);
       mockLocation.addLocations( 110.30841,-6.97261,0,0);
       mockLocation.addLocations( 110.30872,-6.97259,0,0);
       mockLocation.addLocations( 110.30894,-6.97257,0,0);
       mockLocation.addLocations( 110.31006,-6.97253,0,0);
       mockLocation.addLocations( 110.31037,-6.97251,0,0);
       mockLocation.addLocations( 110.31061,-6.97249,0,0);
```

```
        mockLocation.addLocations( 110.31145,-6.97242,0,0);
        mockLocation.addLocations(  110.3121,-6.97237,0,0);
        mockLocation.addLocations( 110.31219,-6.97236,0,0);
        mockLocation.addLocations(  110.3123,-6.97236,0,0);
        mockLocation.addLocations( 110.31239,-6.97236,0,0);
        mockLocation.enabled();
```

How To Update or Upload It Back As Artifact

After you clone this project, made some changes, and then you want to upload it to archiva as an artifact. Add this line on your ramanilocationmanager module, build.gradle :

```
uploadArchives {
    repositories.mavenDeployer {
        repository(url: 'http://team.ujuizi.com/archiva/repository/internal') {
            authentication (userName: 'Your username', password: 'Your Password')
        }
        pom.version = '0.0.3-alpha'
        pom.artifactId = 'location-manager'
        pom.groupId = 'com.ujuizi.ramani'
    }
}
```

You can get your usename and password after registering on http://team.ujuizi.com/archiva/

And to executing the command above. On the top right of your android studio panel, click the gradle button. At this point, you will see the list of gradle from your root project, app, and module. Choose the module (:location-manager) > Task > upload > click the uploadArchives.

Releases

ALPHA
This channel should be tested new functionality on the demo project corresponding library. If new functionality works, then will be merge to BETA channel.

BETA
Tester needs to verify that every feature (old and new) are working. After this BETA testing phase is complete, and all test results prove satisfactory, the corresponding testing issues may be closed one-by-one. With each closed testing issue your personal branch, with each commit, may be merged into the PRODUCTION-channel.

PRODUCTION
This channel mean that it is completely ready for use by the users. Therefore, the bugs reported by our end-users at PRODUCTION releases, needs to be taken very seriously!

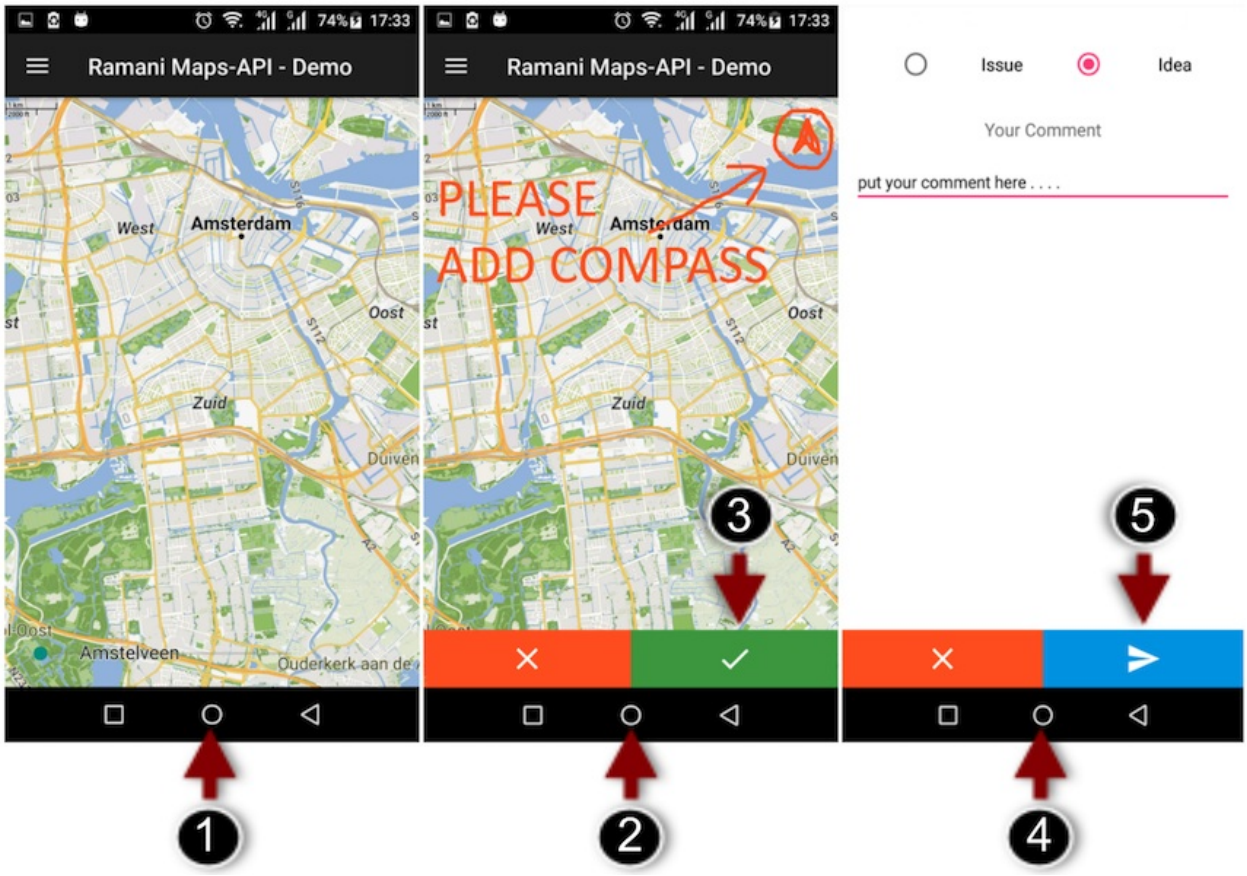
To-Do

- Mock Satellites Data

Contributing

- For testing purposes, users are invited to try early releases from our various release channels, either **Alpha**, **Beta**, or **Production** (depending on how brave you are you can opt-in for our early access releases).
- After installing your release of choice, please provide us with feedback about your experiences (see Feedback)

At any moment in the App, you can submit feedback to report your experiences with the App as follows:



1. Press the Volume-down key, and a Feedback-form will be shown (touch the Back button to revert any time)
2. Draw any on-screen feature to point out wrong or missing elements in the App
3. Touch the OK button to finish your on-screen annotations
4. Use the comment section to provide some textual information clarifying the issue or idea and select the feedback type
5. Touch the Send button to submit

A confirmation e-mail is send for tracking purposes and follow-up correspondence.

License

Acknowledgements

Authors

- Firdaus Kurniawan Zulqornain
- Valentijn Venus
- Firman Wahyudi

Reviewers

- Wahyu Anggara Raya
- Kun Alfin Hidayat

Publisher

- **Ujuizi Laboratories B.V.**