# AR LAB GUIDE

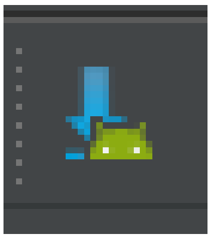
## • Requirements.

Here we have a list of requirements that you will have to prepare at home before the day of the AR lab. It will not take much time to complete. If you have any problem with some step, you can write me an email to: javier.celalopez@heig-vd.ch. I will try to help you and solve it as soon as possible.

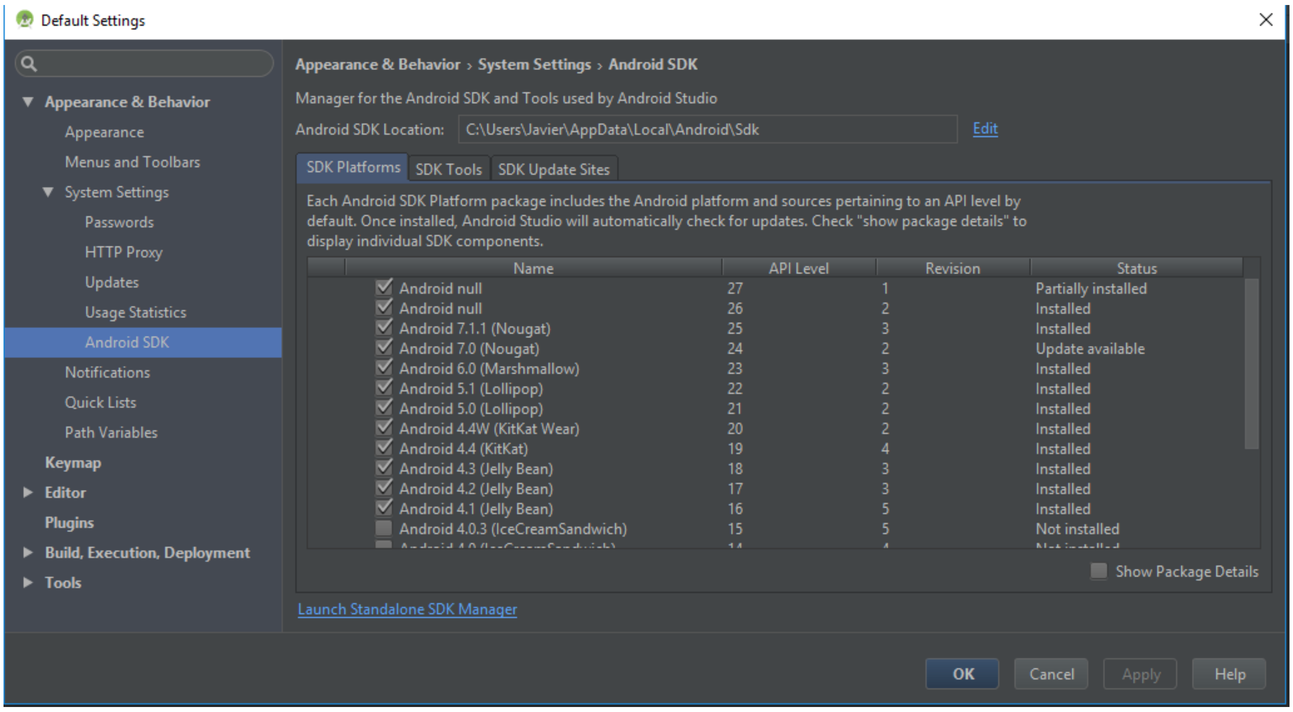
**1. Android SDK installed**. You can install it with or without Android Studio, but probably it's better to do it with it. It's easier to install, and later downloads or updates are managed easier this way. Find the instructions to download it on the different OS here:

https://www.androidcentral.com/installing-android-sdk-windows-mac-and-linux-tutorial

When the installation is completed, make sure to check the versions of Android that you downloaded. Open Android Studio and click on the following icon (upper toolbar; you may need to create a project) to open the SDK manager:



A window like the following should pop.



Default Settings

Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: C:\Users\Javier\AppData\Local\Android\Sdk Edit

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

| | Name | API Level | Revision | Status |
|---|---|---|---|---|
| ✓ | Android null | 27 | 1 | Partially installed |
| ✓ | Android null | 26 | 2 | Installed |
| ✓ | Android 7.1.1 (Nougat) | 25 | 3 | Installed |
| ✓ | Android 7.0 (Nougat) | 24 | 2 | Update available |
| ✓ | Android 6.0 (Marshmallow) | 23 | 3 | Installed |
| ✓ | Android 5.1 (Lollipop) | 22 | 2 | Installed |
| ✓ | Android 5.0 (Lollipop) | 21 | 2 | Installed |
| ✓ | Android 4.4W (KitKat Wear) | 20 | 2 | Installed |
| ✓ | Android 4.4 (KitKat) | 19 | 4 | Installed |
| ✓ | Android 4.3 (Jelly Bean) | 18 | 3 | Installed |
| ✓ | Android 4.2 (Jelly Bean) | 17 | 3 | Installed |
| ✓ | Android 4.1 (Jelly Bean) | 16 | 5 | Installed |
| | Android 4.0.3 (IceCreamSandwich) | 15 | 5 | Not installed |

Show Package Details

Launch Standalone SDK Manager

Make sure that you count, at least, on the version corresponding to your device, if you have one. If not, install those above API level 19. Just check those versions that you miss and click on “Apply”. It may take a while to complete the installation.

**2. Cordova CLI installed.** Follow instructions to install on your OS:

Windows:
https://evothings.com/doc/build/cordova-install-windows.html

Linux:
https://evothings.com/doc/build/cordova-install-linux.html

Mac:
https://evothings.com/doc/build/cordova-install-osx.html

**3. Git installed (with Bash).** Git is used by cordova on the background, so it’s important to have it installed. I know that you have already a command prompt, but we are going to be using Git Bash in order to “unify” commands.

https://git-scm.com/downloads

**4.** As some extra considerations, any text editor can be used for the lab. However, I recommend you to install **Visual Studio** (if you work on Windows; it’s the one that I will be using). If you work on Linux or Mac, as I say, any text editor should do. A good alternative could be **Sublime Text** (simple to use), or even an IDE such as **SharpDevelop** (more complex to use). I leave this to your personal preference.

**5.** If possible, it would be helpful if you could bring your personal Android device (smartphone or tablet). If you don’t have one, don’t worry.

If you do, make sure to enable developer options on your device. It’s done differently for each model, so you can search in google how to do it in yours. Don’t worry because it isn’t difficult at all. For example, in my own device you have to tap repeteadly on a button until a message of activation appears.

When you manage to do so, open “Developer options” in your “Settings” menu and check “USB debugging” and “USB installing”. This way you will grant permission to your computer to install apps on your device via USB. Needless to say, please bring an USB connection as well.

**5.** For developments on iOS, you will need to have a Mac computer with Xcode installed, as well as the packages mentioned in this document. Even though we will not be developing for iOS in this lab, the code is the same. We will see how to prepare a Wikitude project for iOS in the same day of the lab.

- ## Creating our first AR app.

This part of the guide is going to be followed during the lab, so you don't need to do the following things at home.

We are not creating incredible content right now, but the goal of this example is to show how simple creating a simple AR app could be. I have been experimenting with Wikitude and other SDKs and this is the simplest it can get.

- Open a terminal and navigate to the folder where you want to create your app (in my case, ***CordovaProjects***):

```
$ cd ~/CordovaProjects/
```

- Create a project running the following command:

```
$ cordova create ARFirstApp com.javier.arsample ARFApp
```

- Add the platforms that you want to create the app for:

  `$ cordova platforms add android@6.2.3` (depends on version)

  ```
  $ cordova platforms add ios
  ```

- Add Cordova plugin (it takes a while):

  ```
  $ cordova plugin add https://github.com/Wikitude/wikitude-cordova-plugin.git

  cordova plugin add cordova-plugin-compat

  cordova plugin add cordova-plugin-file
  ```

- You can check if everything has been added by typing:

  ```
  $ cordova plugin ls

  $ cordova platforms ls
  ```

You should now include the license key. You will find it in your *guide.txt* file, as well as the places where you need to paste it. It's just a free trial license key, so unfortunately we will have a little watermark on our screen.

When all this is properly performed, you should have a new folder containing an empty project ready to be coded. The structure is quite simple, but for this demonstration we will be focusing on just one part, which corresponds to the folder ***www***. In this folder we will have all the JavaScript, HTML, CSS code and resources, such as images, collections, etc. In the zip file that I provided you, there are some useful files that we are going to need now. Please unzip its content in the ***www*** folder of your project. Let's have a look at it:

- ***jquery***: Here we find the JQuery utils, which is something that I didn't do so we are not going to lose time on it because there's no need for you to know.

- ***js***: Now we have to stop here. This *index.js* file contains functionality provided by Wikitude SDK in order to load AR worlds with this cordova plugin. Same as before, not very important for you to know in depth. But we have this *ARIndices.js* here. Here we are going to define certain parameters needed for loading an AR world. We will create an array of JSON objects which contain this parameters in the way Wikitude needs them, so its very important to name and define things exactly as I am going to do here. Actually, I'm pretty sure that you are already familiar with this kind of structure because you have seen GeoJSON objects if I'm not mistaken.

  We are going to create this variable named "worlds", which is going to be this array that we were talking about. In order to define the first world's parameters:

  ```
  var worlds = [
     {
        "path": "www/world/1st/index.html",
        "requiredFeatures": [
           "geo" // We could have image_tracking too
        ],
        "startupConfiguration": {
           "camera_position": "back", // back camera
           "camera_resolution": "auto" // automatic will do
        }
     }
  ]
  ```

  Now, our variable world contains the parameters needed for Wikitude to load the first world. Of course, we have to actually create this first world, but that will come. In this same file, we have to create a way of accessing this array of JSON objects. This  very simple function will do:

  ```
  function getWorldPath(index) {
     return worlds[index];
  }
  ```

  This could be done in another way, but it's easier and more practical to keep it as if it were a "private class" in a separate file.

- ***worlds***: Here it's were we are going to actually create the AR environments, worlds, whatever. As you can see, I've provided you with the folder containing the structure of the first world, so this way in order to create another one you just have to copy it and change its content. As I told you, this structure things are not that important and it's not what you are supposed to learn in this lab. Okay, so having a look at its content, we find the folder ***img***, where we are going to find all the resources that we are going to need for this tutorial. Basically, Here we will have the images that we will display on the AR view, as well as the marker images that we could need. In this case, no need for that. Also, 3D models will be included in this folder. Same here for JQuery library. The *index.html* is going to have some not very useful thing but kind of helpful. Don't bother to try to understand this because it actually could be removed and nothing would happen. I will use it to show you some things while the app is running. Finally, the ***js*** folder will containf the definition of the world. We will start coding now and learning to create AR content. Don't be afraid, you will realise how simple it is to introduce yourself in AR developments. [Note for professor: it could seem like suddenly there's a lot of code, but I will write it at the same time as them and explain everything little by little so no one gets lost]

```
var world = {
    loaded: false, // Checks if world has been already loaded
    loadPois: loadPoisFn(poiData) {
        var image = new AR.ImageResource('img/marker.png');
        var imageDrawable = new AR.ImageDrawable(image, 2.5, {
            opacity: 0.9 // Doesn't matter actually
        });
        var location = new AR.GeoLocation(poiData.lat,
        poiData.lon, poiData.alt);
        var geoObject = new AR.GeoObject(location, {
            cam: [imageDrawable]
        });
    },
    locationChanged: function onLocationChanged(lat, lon, alt,
    acc) {
        if (!world.loaded) {
            poiData = {
                "lat": lat + (Math.random()/5 - 0.1), // randomness
                "lon": lon + (Math.random()/5 - 0.1),
                "alt": alt
            };
            world.loadPois(poiData);
            world.loaded = true;
        }
    }
}
```

So, let’s have a look at this that we have here. We added some randomness to the POI in order to see it without having to move. We set altitude to 100 because, as you will see, this random inclusion can make the point be realy far. We will change this later. In order to run our application and make it work, there is still something to do. We have to make sure that this function is triggered when our location changes. This event is going to be triggered by Wikitude, and we should assign it this function as event listener at the end of the document:

```
AR.context.onLocationChanged = world.locationChanged;
```

Again, make sure to use the exact same names. I’m going to include a little code here in order to now where the point actually is. You don’t need to do it:

```
console.log(‘alt: ’ + poiData.alt + ‘ lon: ’ + poiData.lon);
```

We will now run our application and see what happens. Save all this files and type in the terminal the following command:

```
$ cordova run android
```

It will take a while to build, and especially the first time, so be patient with it.

After building the application, two things can hanppen. First option, it runs correctly and you can test it on your device (being realistic, it’s not going to happen). Second option (most likely), some errors appear during building. If it stops bulding throwing some Java exceptions, try to check your internet connection because it’s going to be that for sure. If it stops with some message similar to:

```
* What went wrong:
    A problem occurred configuring root project 'android'.
        > You have not accepted the license agreements of the
        following SDK components:
        [Android SDK Platform (number)].
        Before building your project, you need to accept the
        license agreements and complete the installation of the
        missing components using the Android Studio SDK Manager.
        Alternatively, to learn how to transfer the license
        agreements from one workstation to another, go to
        http://d.android.com/r/studio-ui/export-licenses.html
```

it’s because you don’t have the target version or you need to accept its license. You have three possible solutions:

- Launch Android Studio and open SDK Manager. Install or update the number of version that the error shows.

- Open your terminal and go to your Android SDK home directory (on windows, generally ***C:\Users\*USERNAME*\AppData\Local\Android\sdk\***) and navigate to folder ***/tools/bin/***. By typing `./sdkmanager --licenses`, all the remaining licenses to accept will appear and you can accept them one after the other.
- If none of the above works, go to your project directory and open the file ***/platforms/android/****AndroidManifest.xml*. You will see an attribute in label `<uses-sdk>` similar to:

  ```
  […] android:targetSdkVersion="25"/>
  ```

  In my case, it is set to 25. Downgrade this value for the one according to your highest Android SDK version (you can see it in the SDK manager).

When this problem is solved, you can try and rerun the application. It could happen that it doesn't launch it yet. Now, the problem coul look something like this:

```
Manifest merger failed:
uses-sdk: minSdkVersion 16 cannot be smaller than version 19
declared in library
```

It's easy to notice that the problem here is that your minimum version of Android is lower than the smallest version that Wikitude requires, which is 19. The solution is very simple. In the file ***/platforms/android/****AndroidManifest.xml*, (same exact line as before), change the `minSdkVersion` attribute of label `<uses-sdk>` and set it to 19.

(Open ***chrome://inspect*** in Chrome) This is the Chrome's tool for debugging these applications. If you remember de "console.log" we wrote before, this is where it's going to act.

(When it runs) So well, if as we have introduced some randomness, we will have to look for the marker. (Find it and show it) This is the simplest way of overlaying augmentations on geographical points of interest. This is the starting point for everything you could imagine using this technology. We defined a geographical location and we displayed an image on it. We can actually see where the point is in Google Maps. We simply type its coordinates and we can see that, due to the randomness that we put, this point is in (whatever), which is actually quite far for us now. If we would like to somehow interact or modify the augmentation we should define a close point.

For example, we cannot appreciate here because the point is far away, but I can assure you that this image is going to stay the same size no matter how much you approach. This is one of the first principles of augmented reality which we can actually read in (take book and show it). To create this "sense of immersion", it is mandatory to take care of aspects such as the size of the object according to the distance to it (in real life, when you walk away from the table, you perceive it smaller, and the other way round; I mean, this is not a secret to any of us and it's not necessary to read a book and to study a lot to notice this). Another parameters could be its opacity, its orientation depending on the perspective, etc, and this is exactly what we are going to work here today. As we are working with 2D augmentations for the moment, we are going to start with this size issue. Let's start by defining a closer location. We can do it by simply pointing it in Google Maps and writing down its coordinates.

Okay, let's think about it. In this very moment, size and distance are totally independent variables. What we want to do is to make the object grow as we approach it. This is equivalent to say that the size of the object is going to increase when the distance from it to the user decreases. It's easy to notice that in this approach we the same two variables here, but now one dependent variable, size, and one independent variable, distance. The relation between these variables is going to be inverse (this is totally equivalent to what we said before). We have to decide now the function to describe the size of the object in terms of the distance. First of all, the size of the drawable is defined in the options parameter of the constructor, an it's called scale. If its scale is 1 by default, a new scale of 2 is going to mean that its size is doubled. It could seem trivial, but it's important to understand this.

As relationship is inverse, we could simple use an inverse function (write it in the blackboard) with some multiplier to define the number of meters at which the object will have scale 1 ("default size"):

$$scale(distance) = m \cdot \left(\frac{1}{distance}\right)$$

This would look like (draw it) something like this. Does it seem okay to you? I can find some defects however. Does someone have an idea of problems that it may have? (If they don't) Well, to begin with the limit when distance approaches 0 is infinite, which is not convenient for what we want. The size has to stop growing at some point and, of course, a "reasonable" point, not infinite. This is easily solved though. We could simply just shift the curve by adding something down here (*distance + 1*, draw it; mention not to worry about left part of Y axis: no negative values for X). This way, the size will be limited when distance is 0, which is the lowest point where it can get. Actually, the scale when distance is 0 is going to be? (*m* = meters multiplier) Solved this problem, any other problem to take care of? (If not) Well, it is true that this inverse decreasing here doesn't seem kind of "natural" to the human vision. When it's really close, it grows very much faster that when its not that close. This has a solution as well, but it is not that easy. I would like you to think about it. On the ohter hand, it has its advantages. Of course, this asynthotical approximation of the scale to 0 makes this value impossible to reach, so the object will never disappear entirely from our field of vision. This, let's be practical, it's useful and it's not going to be a problem regardind the distances that we are going to manage. Therefore, we'll keep it as an advantage. It also prevents the scale to reach negative values, which nevertheless could be easily solved as well.

I will show you a way of coding and assigning this function to the actual scale of the drawable. First, we should define some objects as a world variable, as we have to address them from another function (maybe mention how to calculate distance given coordinates in degrees).

```
var world = {
    imageDrawable: null,
    distance: null,
    ...
    loadPois: loadPoisFn(poiData) {
        world.imageDrawable = new AR...
        world.distance = new AR...
    ...
    changeSizeInverse: function changeSizeInv() {
        var dist = world.location.distanceToUser();
        var scale = 10/(dist + 1); // 11 meters = scale 1
        world.imageDrawable.scale = scale;
    },
    locationChanged: function onLocationChanged(...) {
        if (!world.loaded) {
            ...
        }
        world.changeSizeInverse();
    }
}
```

As we are in the classroom, it's difficult to test it by ourselves now with so much people and everything, so I will show you some video of my personal tests on this approach. (Comment video).

Any other proposal for scale function? We could simply have a decreasing linear function (draw it) with some parameters such as the slope (determining how fast it's the scale going to decrease) and point where it cuts the Y axis (determining the maximum size/scale, given when distance is 0).

$$scale(distance) = s * distance + maxscale$$

The problem here is that we can find negative values, so we have to cut the line when size is 0. This approach seems a more "natural" way of representing scale as a function of distance. However, it counts on the disadvantage of making the image disappear at some point. What should we do here? Cutting a little higher than 0? Defining a piece function containing these two approaches? I would like you to think about it as well. Anyway, we could define it this way:

```
changeSizeLinear: function changeSizeLin() {
        var dist = world.location.distanceToUser();

        var scale = -0.1*dist + 10;

        world.imageDrawable.scale = scale;

}
```

and, of course, we’ll have to call it instead of the inverse one. We see here another video corresponding to this approach.

Now, having listed every advantage and disadvantage, what would you do? (Draw piece functions in the blackboard). This is going to be part of the exercises that you are going to do at home. Not only you will define this dynamically changing size, but also some other parameter that could be interesting to create really “immersive technology” (cite here wikitude documentation).

(If possible, introduce 3D model augmentations and make the exercise with them)