Connecting an IMU device over BLE to an iPad

Joshua Karch
September 16,2021

Introduction

I had developed, using Altium® Designer a PCB with an inertial sensor and Bluetooth Low Energy interface. That PCB is fairly simple, containing an ESP32 microcontroller module with Bluetooth Low Energy (BLE) support, a BNO085 IMU sensor, a boost converter and LiPo battery management system. The precise schematics and layout of this hardware will not be disclosed in this document, and is not the subject of this project brief, however a similar device can be made from four COTS components readily available from Adafruit (www.adafruit.com) or SparkFun(www.sparkfun.com). Connect an ESP32 Feather board (https://www.adafruit.com/product/3405) , a Powerboost 1000 board to charge the LIPO and supply power to the ESP32 Feather (https://www.adafruit.com/product/2465), a LiPO battery (https://www.adafruit.com/product/2011), and a BNO085 IMU board (https://www.adafruit.com/product/4754) together to create a module.

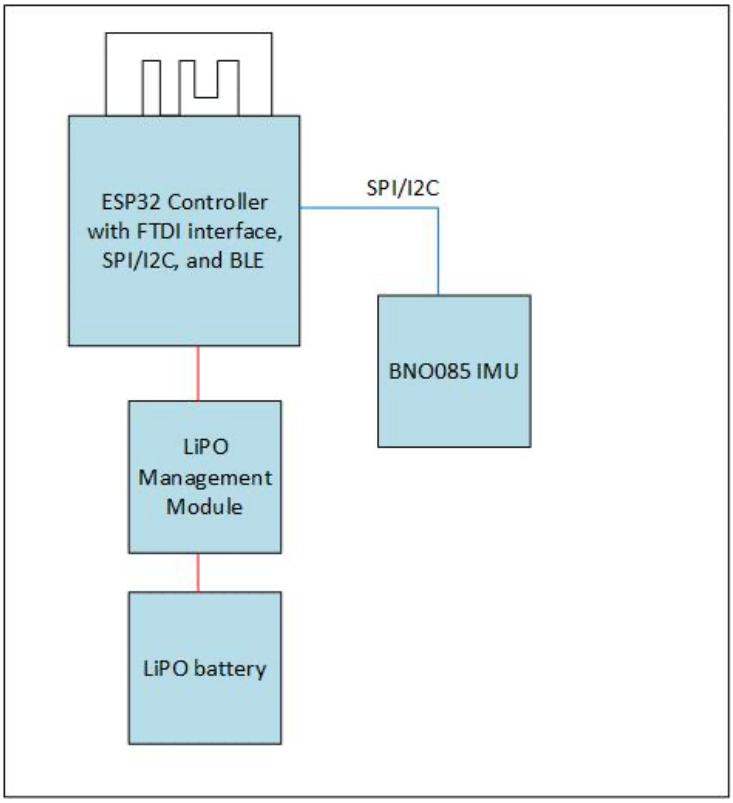


Figure 1: General architecture of IMUBox

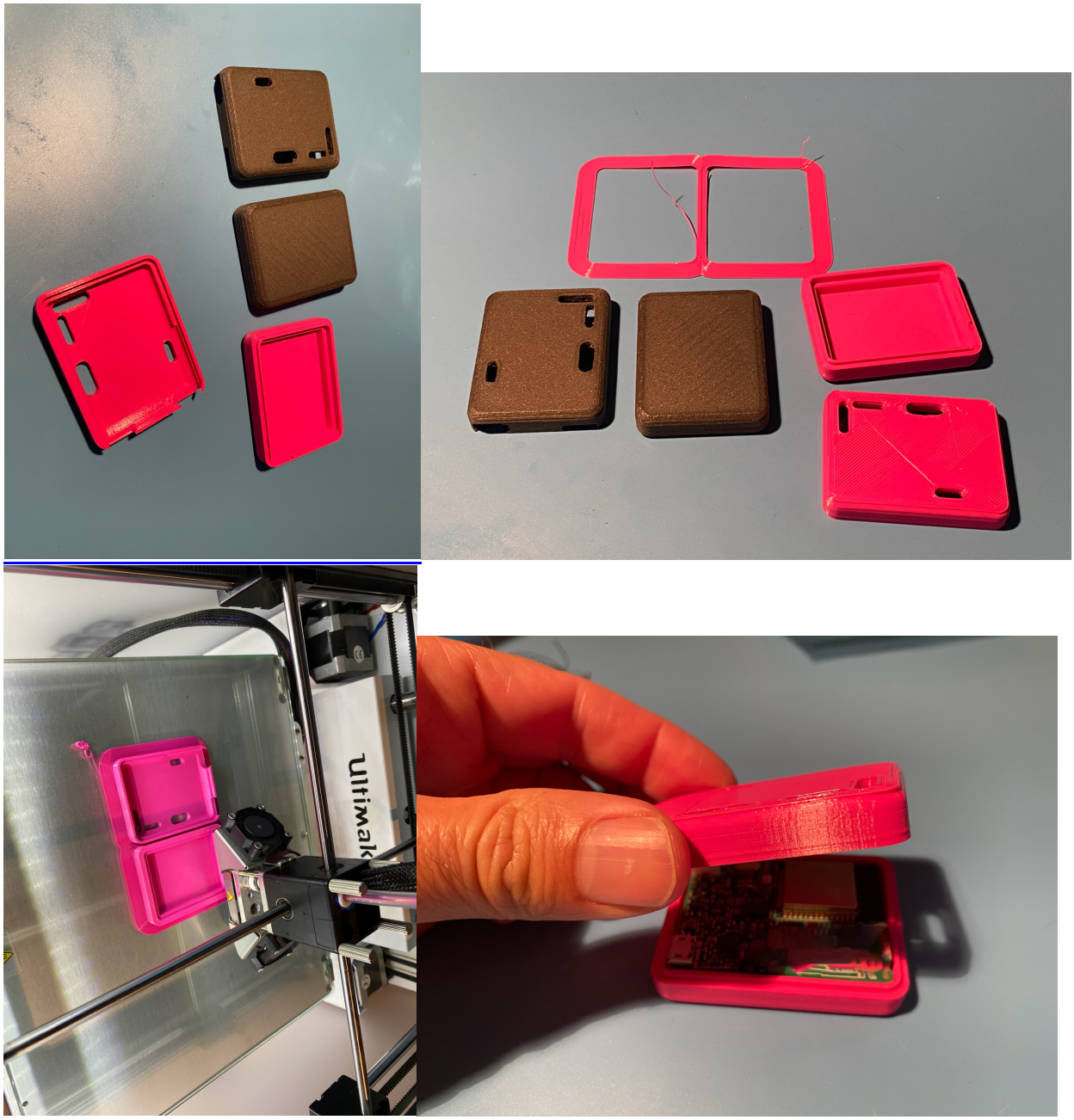


Figure 2-5: Enclosure and PCB (obfuscated)

I decided a few weeks ago to give IOS development a try after many years of reticence, and to get back into 3D printing. I set out with a mission to create an application template that could communicate with a generic device over Bluetooth Low Energy, and to also teach myself Solidworks® along the way. So this document is the culmination of a few days of putting everything together. The software components accompanying this document are located at: https://github.com/jkarch/IMUBox

ESP32 Bluetooth Low Energy IMU sensing application

I decided, for simplicity to write the IMU sensing application using the Arduino environment, updated for ESP32 support. Download Arduino from Arduino.cc or directly from Arduino or Windows store. Once downloaded, proceed to launch the application and go to Arduino Preferences.
Add the espressif content in order to download the Board Support Package for Arduino. More info can be found here: https://dronebotworkshop.com/esp32-intro/

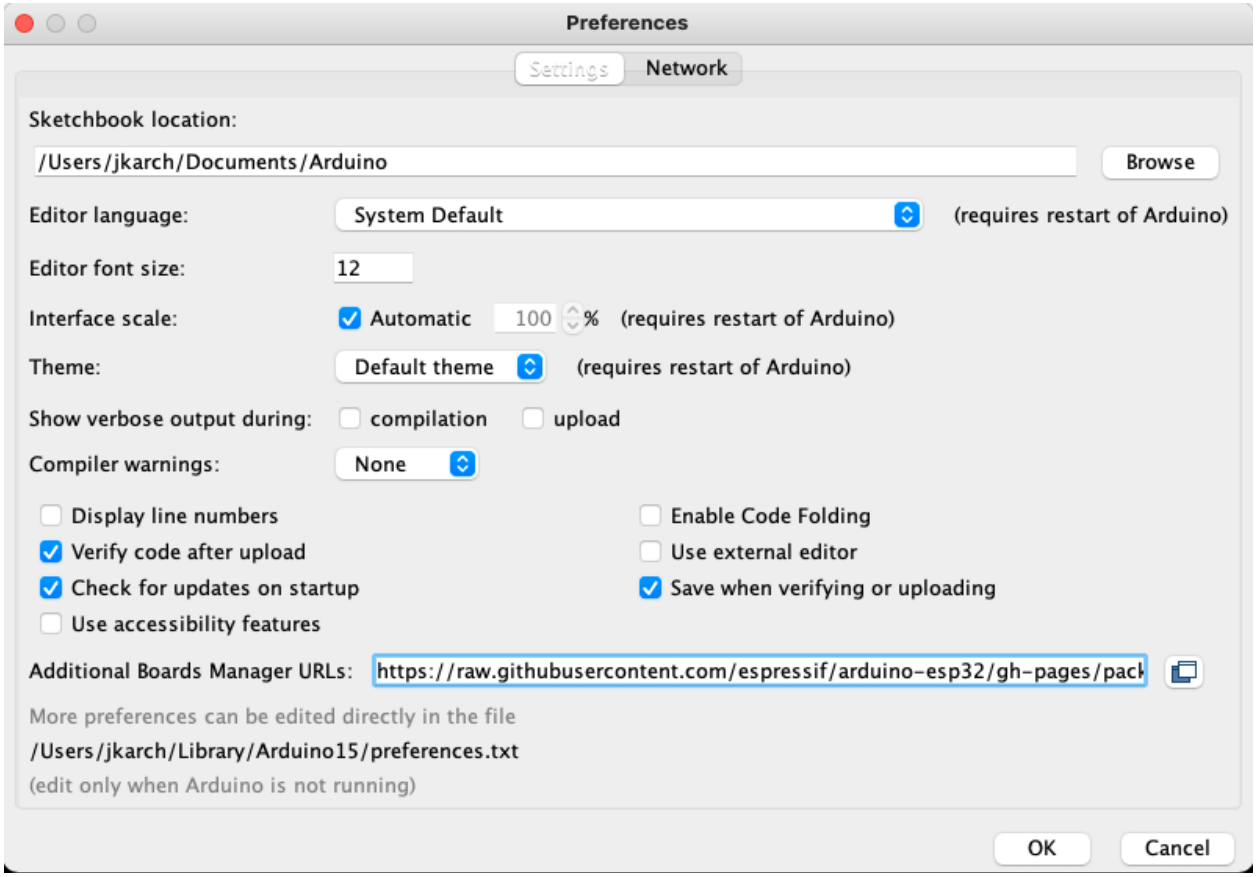


Figure 6: Arduino Preferences

Enter the following in the Additional Boards Manager URLs:
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
Under Tools:Manage Libraries search for BNO080 and install the Sparkfun BNO080 Cortex Based IMU library.

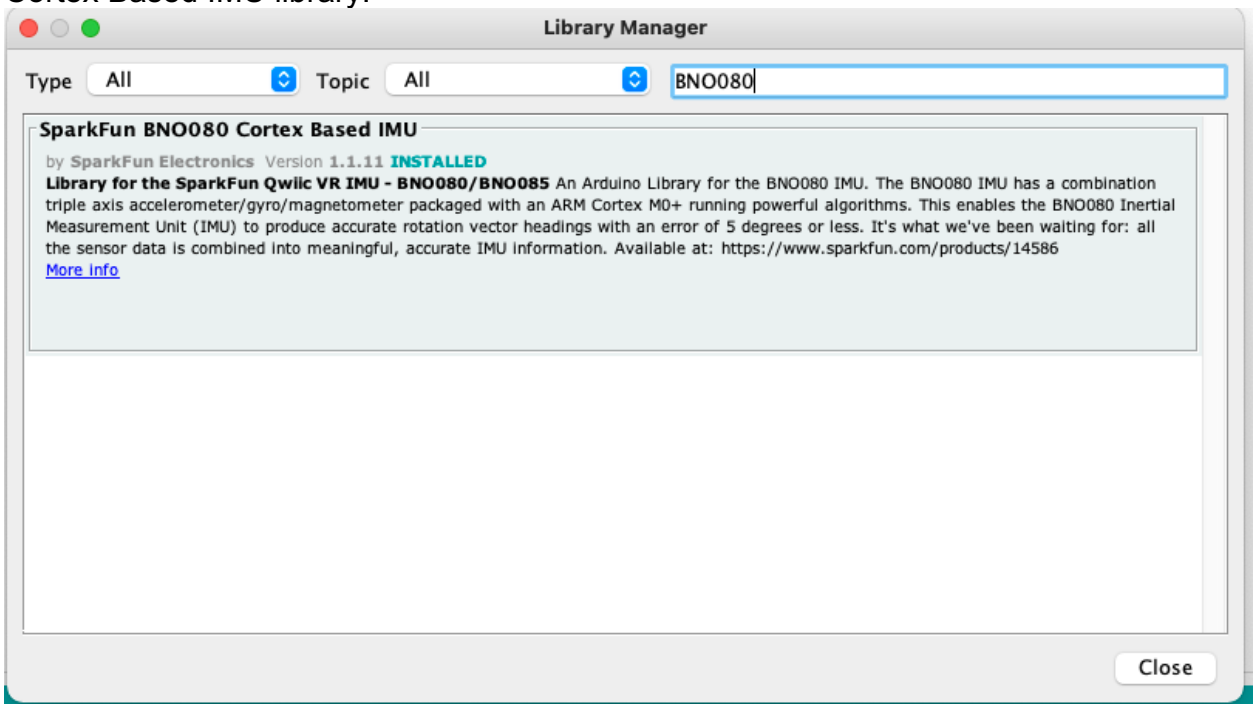


Figure 7: Install libraries

Under Boards, Board Manager search for ESP32 and select the ESP32 package:

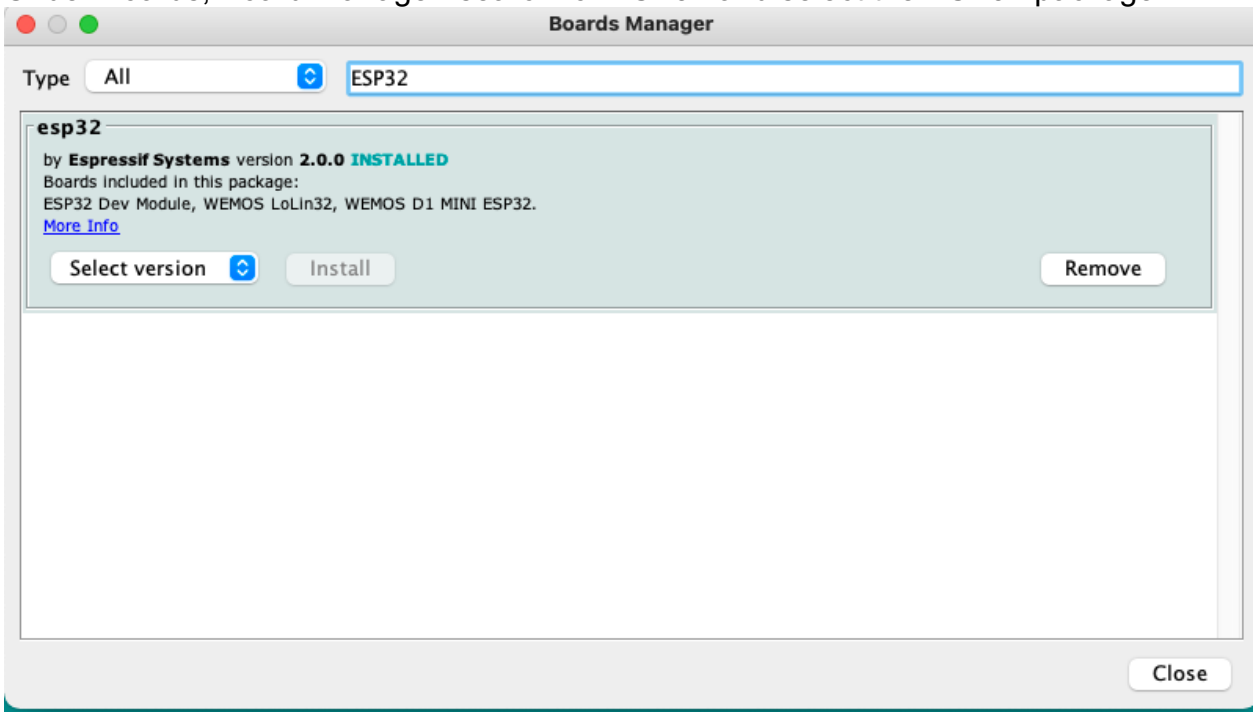
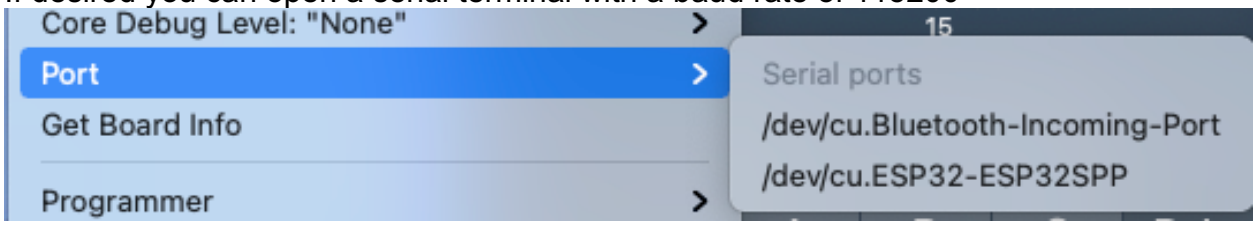
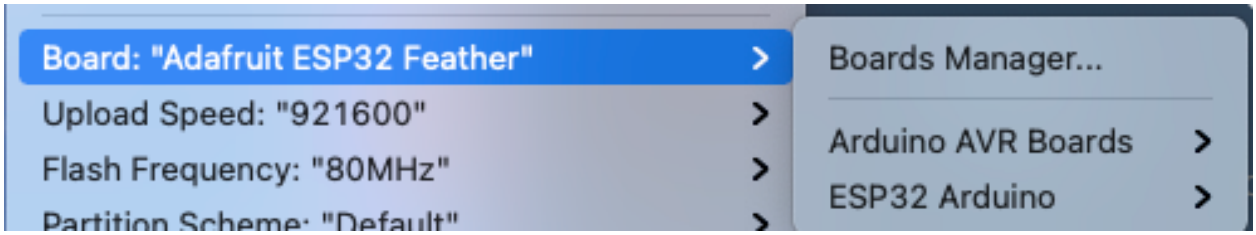


Figure 8: Add Board Support Package

If desired you can open a serial terminal with a baud rate of 115200



Load imubox.ino from Github at:
https://github.com/jkarch/IMUBox/tree/main/imubox_esp32_arduino
and select Adafruit ESP32 Feather



Click  to compile and upload to the ESP32 Feather.

If successful, IMUBox will appear as a device on LightBlue LE and begin streaming data. Now you can move to the application side: But before that, a little bit of context. This Application was written using a build of FreeRTOS to manage the task of receiving data from the BNO08x sensor, which is interrupt driven, to reading that data in a task that is locked by semaphore blocking for new data, and then into a queue which takes every fourth sample and transmits over BLE. Meanwhile, another low priority task checks for BLE device disconnects, and restarts the advertising processes. An additional callback looks for communications from a device and can process messages or commands, but this is not implemented beyond handling received data and doing nothing. This application is intended to be a simple template for processing data from multiple sources using an RTOS. One other thing to note is you should generate your own UUIDs for BLE Services and Characteristics for a unique device that you create. https://www.uuidgenerator.net

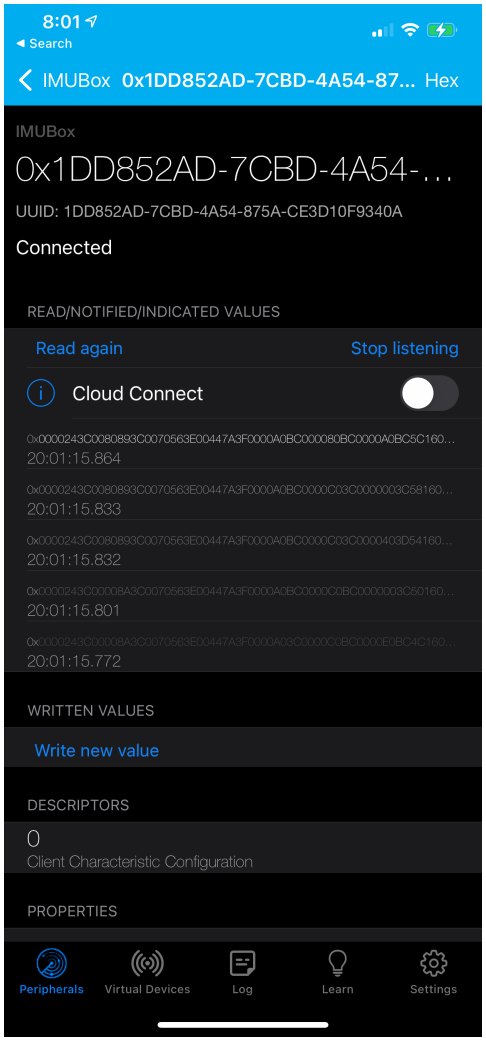


Figure 9: Here LightBlue shows data streaming from the IMUBox device

iPad IMU Application

The iPad Application is my first foray into iOS apps, and the first time I've communicated with BLE with an iOS device. While streaming data from an IMU is something that preferably should be done with Bluetooth Classic, the level of effort required to qualify a Bluetooth Classic device for use with iOS is considerable. That being said, a Bluetooth Serial Port can run a lot faster, without the hiccups of Connection Intervals. For more info on Connection Intervals, go to: https://www.novelbits.io/ble-connection-intervals/

Getting Started: Before going any further, I want to give credit to my sources, since I was able to quickly get up and running with the help of these sites. I decided to use AR with RealityKit and Swift UI, because I could import a 3D model of the enclosure of my device, project it in an environment, and compare it to the IMUBox device.

Figure 10: The app.

Note—this is a very new field for me, so if anyone has suggestions on how I can improve the architecture, please provide feedback. That being said, for starters it appears I got it working: Feel free to send me a message at jkarch@eyebotix.com

For the very basics of Bluetooth Low Energy with IOS, and getting started with the Apple Developer Membership, check out Adafruit's blog. I did not use this in making my app because Apple has switched to SwiftUI, and also the application was a bit too simple (it sends and receives text messages) so it wasn't able to handle my main interest of receiving streaming sensor data. https://learn.adafruit.com/build-a-bluetooth-app-using-swift-5

I used the following site to get an @EnvironmentObject data control model working. https://medium.com/twinkl-educational-publishers/create-your-first-ar-app-with-realitykit-and-swiftui-7c5d1388b5 An EnvironmentObject allows the easy sharing of data among multiple SwiftUI panes. I would follow this guide to set up a RealityKit app first, before delving into Bluetooth Low Energy communication. Note that when you first create a template project as per https://medium.com/twinkl-educational-publishers/create-your-first-ar-app-with-realitykit-and-swiftui-7c5d1388b5, build the app to get rid of some initial bugs before you even start coding.

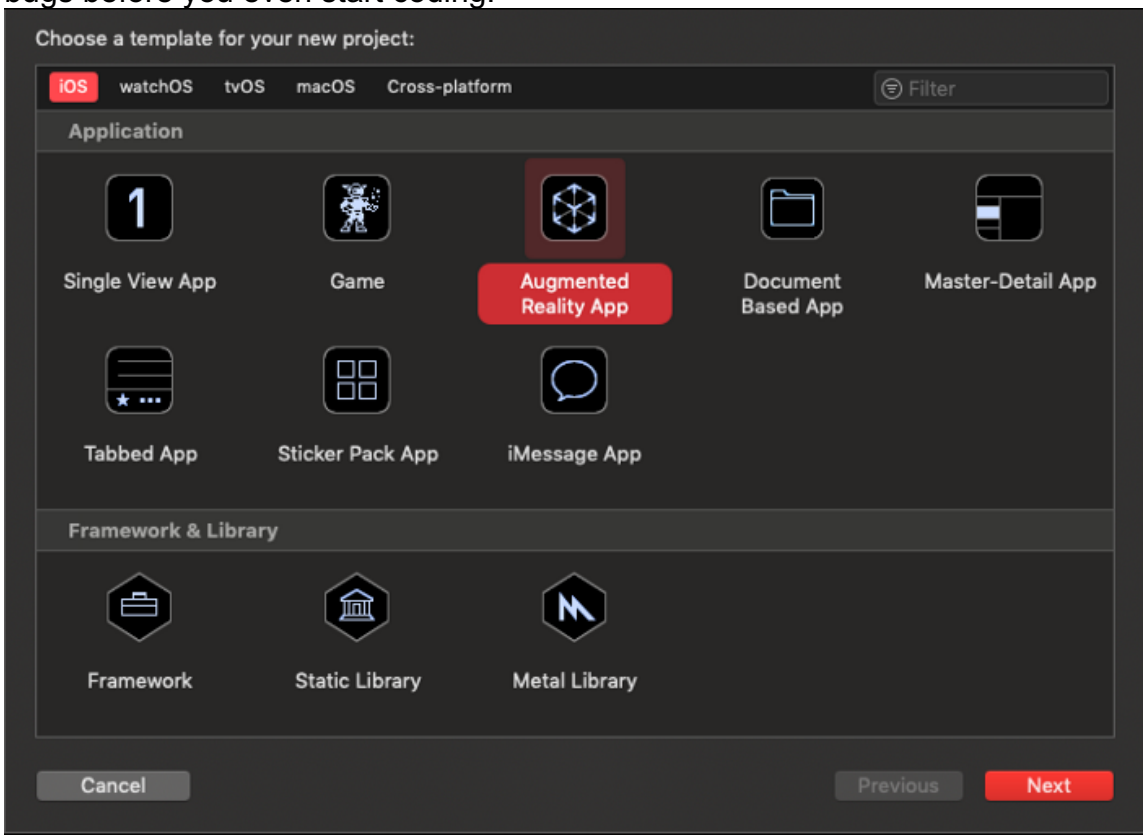


Figure 11: Select Augmented Reality App

I derived MyBLEManager.swift from https://github.com/paul-ios-dev/SwiftUI-BLE-AAR/tree/master/SwiftUI-BLE-AAR and added in code to process byte data read over BLE. Once connected to a device the main function of interest to modify in that file, besides defining a struct called IMUSamples, is to do an "unsafe" byte copy of each section into the struct.

```
public func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {

  if characteristic.uuid.uuidString == "1DD852AD-7CBD-4A54-875A-CE3D10F9340A" {
    let mydata = characteristic.value!
    for i in 0 ... 31 {
      print("\(i):\(mydata[i])", terminator:" ")
    }
    print("")


      self.IMUSamples.Qx=mydata.withUnsafeBytes { $0.load(as: Float.self) }
      self.IMUSamples.Qy=mydata.withUnsafeBytes { $0.load(fromByteOffset: 4, as: Float.self)}
      self.IMUSamples.Qz=mydata.withUnsafeBytes { $0.load(fromByteOffset: 8, as: Float.self)}
      self.IMUSamples.Real=mydata.withUnsafeBytes { $0.load(fromByteOffset: 12, as: Float.self)}
      self.IMUSamples.Ax=mydata.withUnsafeBytes { $0.load(fromByteOffset: 16, as: Float.self)}
      self.IMUSamples.Ay=mydata.withUnsafeBytes { $0.load(fromByteOffset: 20, as: Float.self)}
      self.IMUSamples.Az=mydata.withUnsafeBytes { $0.load(fromByteOffset: 24, as: Float.self)}
      self.IMUSamples.SampleCount = mydata.withUnsafeBytes { $0.load(fromByteOffset:28, as: UInt32.self)}

      print("pkt size: \(mydata.count)")
      let ssize=MemoryLayout<IMU_Payload>.size
      print("struct size: \(ssize)")


  }
}
```

Figure 12: the core part of *myBLEManager.swift*

At the time, I could not find a way to simply memcpy and cast bytes into a struct, because the size of Swift struct contains additional padding. I welcome feedback on how to do something like
memcpy(&IMUSamples, databytes, sizeof(databytes)/sizeof(databytes[0])

But for now, this works, it's just more wordy and explicit. Also notice there is no error checking for smaller packet sizes. That's something that should be added, but for the purpose of this test isn't an issue. I took the *MyBLEManager.swift* file and turned it into an *EnvironmentObject*, and linked it in the *AppDelegate.swift* file

```
// Create the SwiftUI view that provides the window contents.
let contentView = ContentView()
    .environmentObject(DataModel.shared)
    .environmentObject(BLEConnection.shared)
```

The other area that's important to notice is the *info.plist* file. This is where you instruct the iOS device that Bluetooth will be used, even in the background. Without it, Bluetooth Low Energy can't be used and the app will crash:

I added three keys to the list:

Key	Type	Value
Information Property List	Dictionary	(18 items)
Localization native development region	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle version string (short)	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Privacy - Bluetooth Always Usage Description	String	Bluetooth in use with this app
Privacy - Bluetooth Peripheral Usage Description	String	Can this app communicate with Bluetooth peripherals?
Privacy - Camera Usage Description	String	
Application supports indirect input events	Boolean	YES
Required background modes	Array	(1 item)
Item 0	String	App communicates using CoreBluetooth
Required device capabilities	Array	(2 items)
Item 0	String	armv7
Item 1	String	ARKit
Status bar is initially hidden	Boolean	YES
Supported interface orientations	Array	(3 items)
Supported interface orientations (iPad)	Array	(4 items)

Figure 13: *info.plist*

In order to deploy to an actual iPhone/iPad, you need to set up signing and capabilities and get an account at Apple for XCode. The project root contains the project settings:

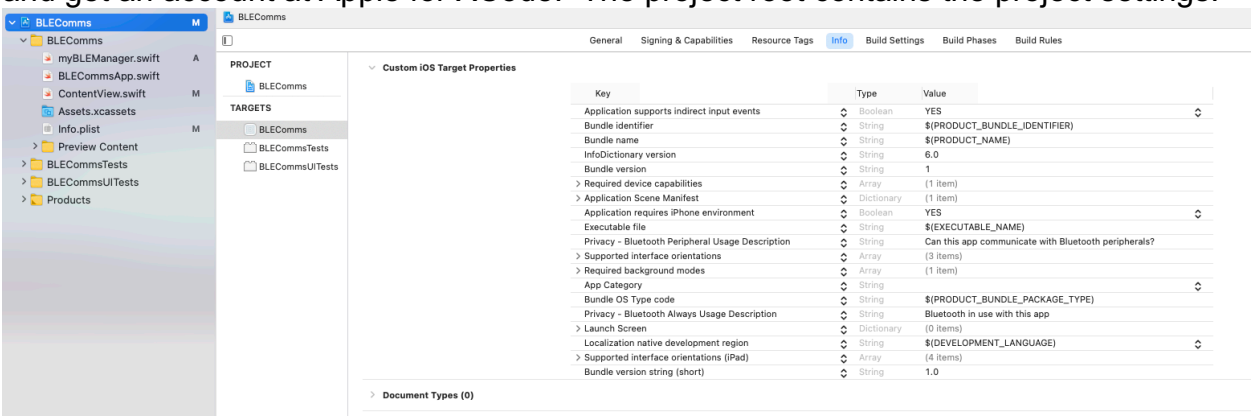


Figure 14: Project Settings.

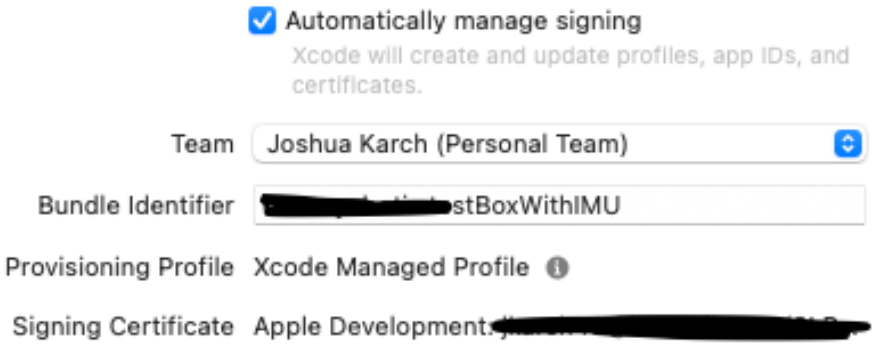


Figure 15: team connection information.

If you've gotten this far and merged the AR application from the medium.com demo, with SwiftUI BLE-AAR, the one thing that's not covered yet is replacing the cube with a model of the enclosure used. I took an STL model of the enclosure and opened it up in XCode, moved the item into place to align it, and saved the file as a ".usd" file, which is a Pixar format:

Figure 16: The STL file that gets converted to a .USD file after being zeroed in position.

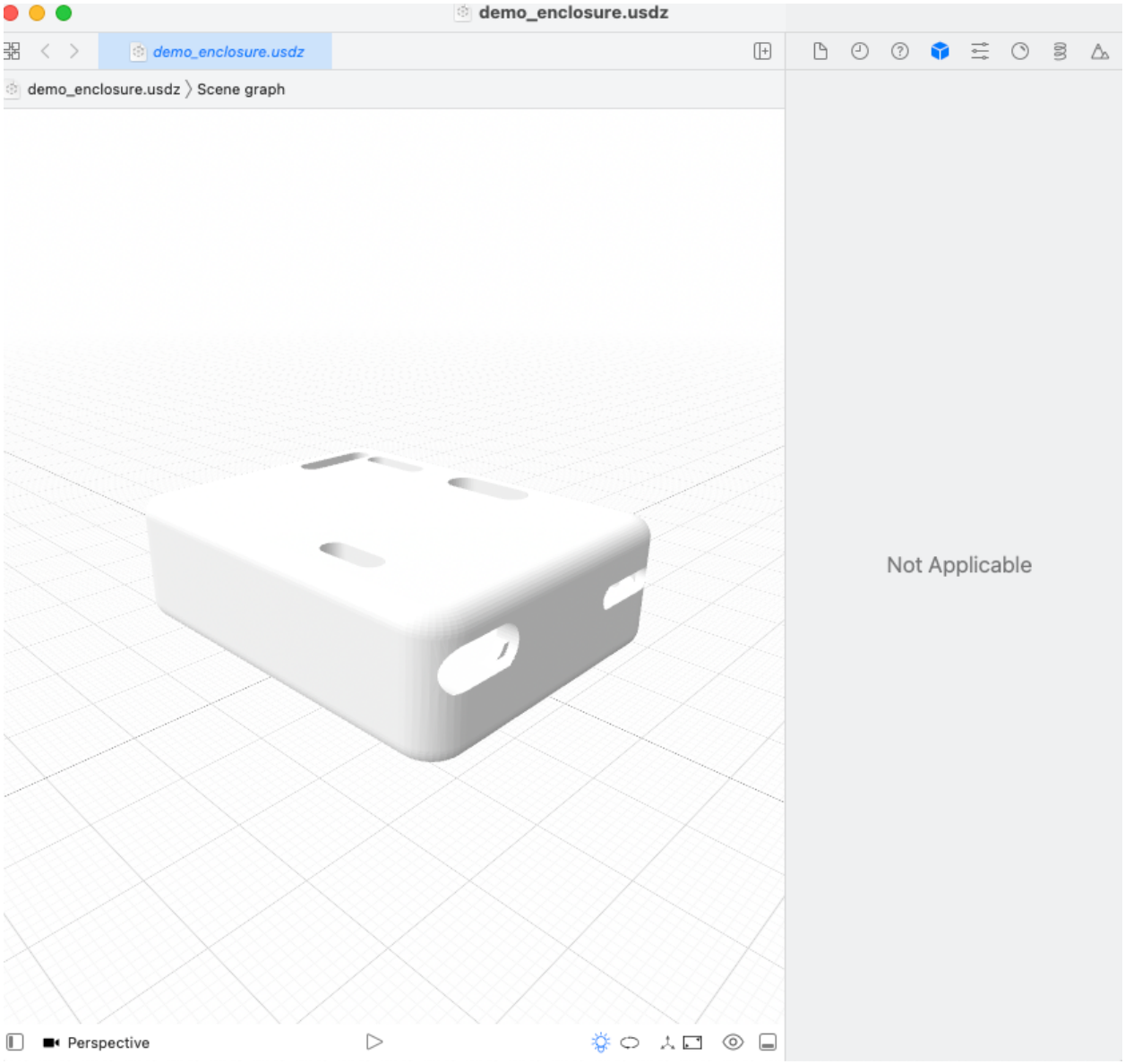


Figure 17: the STL file which gets converted to a USDZ file.

I downloaded “Reality Converter” https://developer.apple.com/news/?id=01132020a Which basically allows you to open a USD file an create a USDZ file. The main difference between the files is that you can open a USD file and apply colors or textures. Then take a Jpeg swatch of the color and apply it and re-save the file as a USDZ.

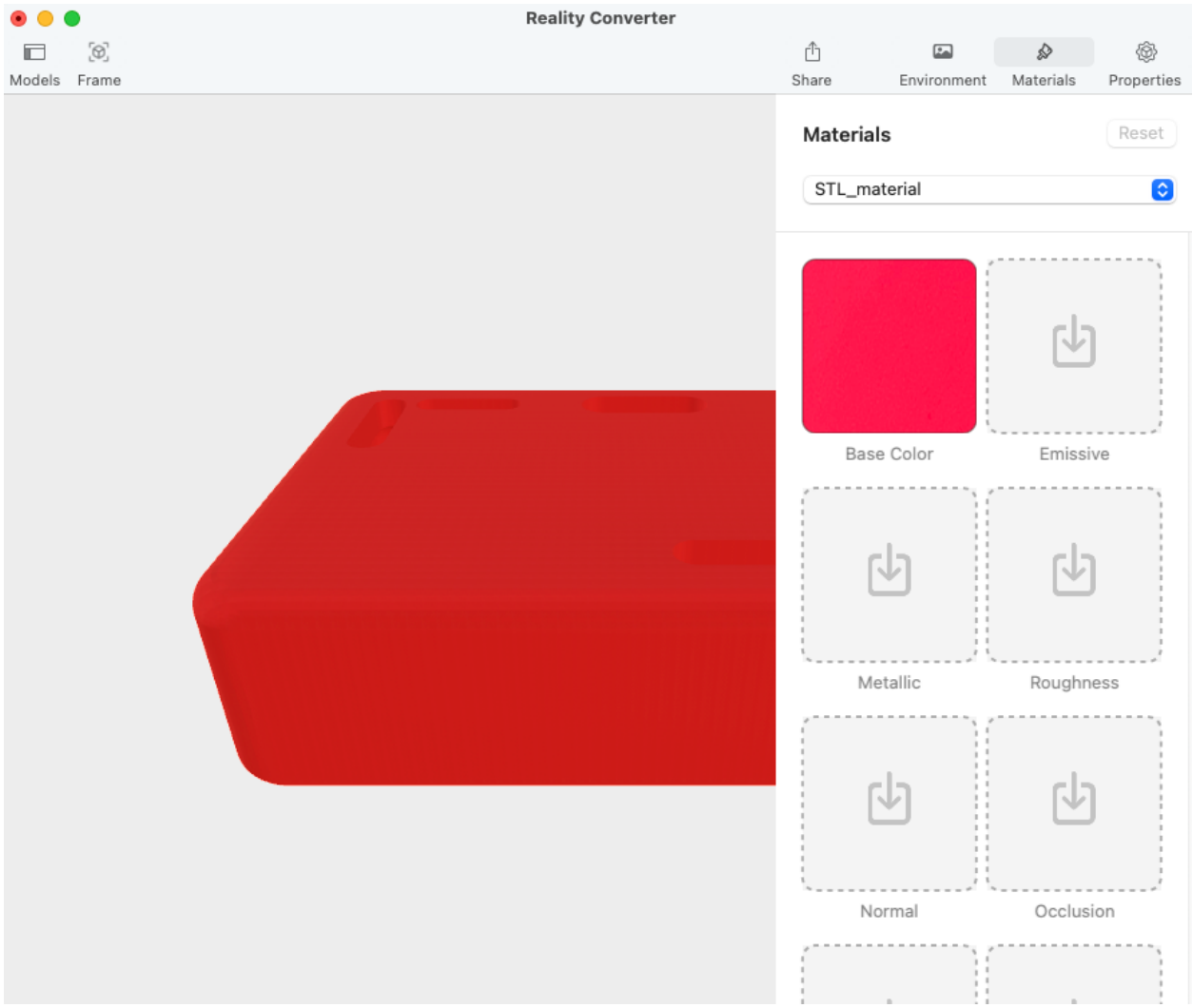


Figure 18: Reality Converter

Within XCode, click on Experience.rcproject, then on the right hand corner, select "Open in Reality Composer."

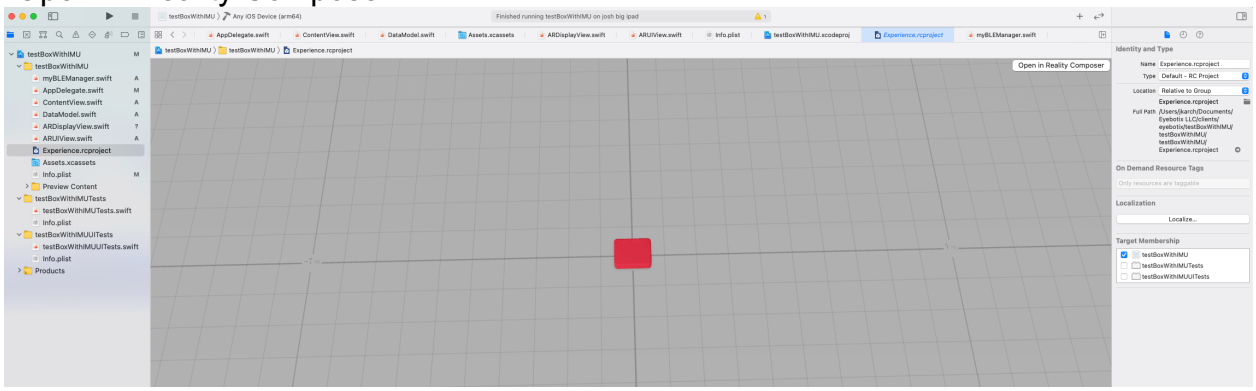


Figure 19: Reality Composer

Once in Reality Composer, go to + to add your library component, then drag to the zero axis. Note the item might be quite large and you have to zoom out, then scale the object smaller.

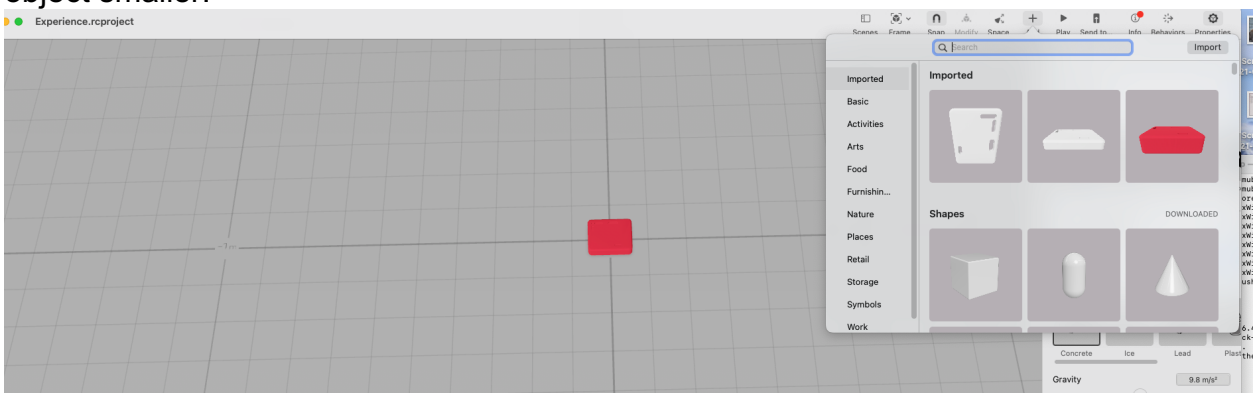


Figure 20 Adding library component.

Set the Scale and transform and name it imuCase, as in this example, in the text box above “Transform.”

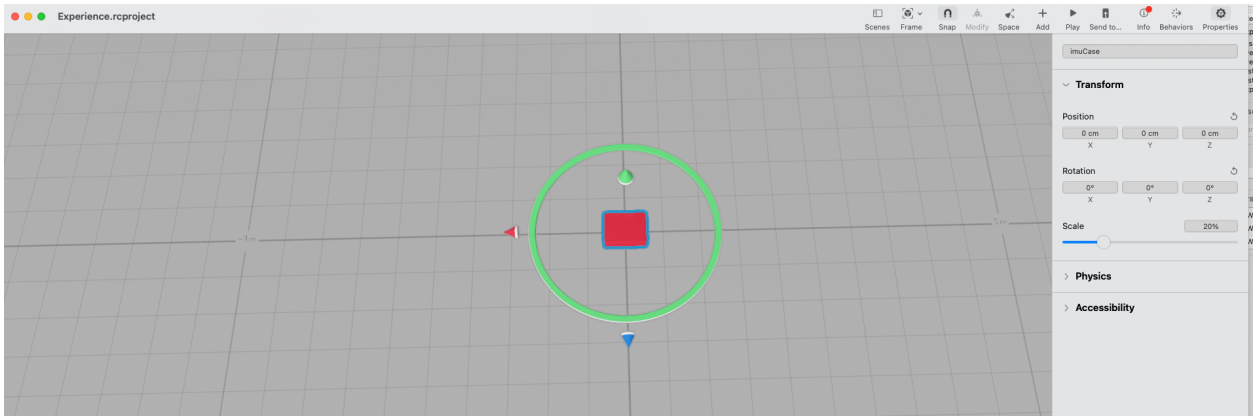


Figure 21: Align and scale the object and name it.

For more information on USDZ, visit For more info, check out: https://usdzshare.com/reality-kit/

Close, save, and go back to XCode and save the project.

From there to deploy to a device. If your iPhone is connected to your computer, unlock it, select iPhone, and click build, then run.

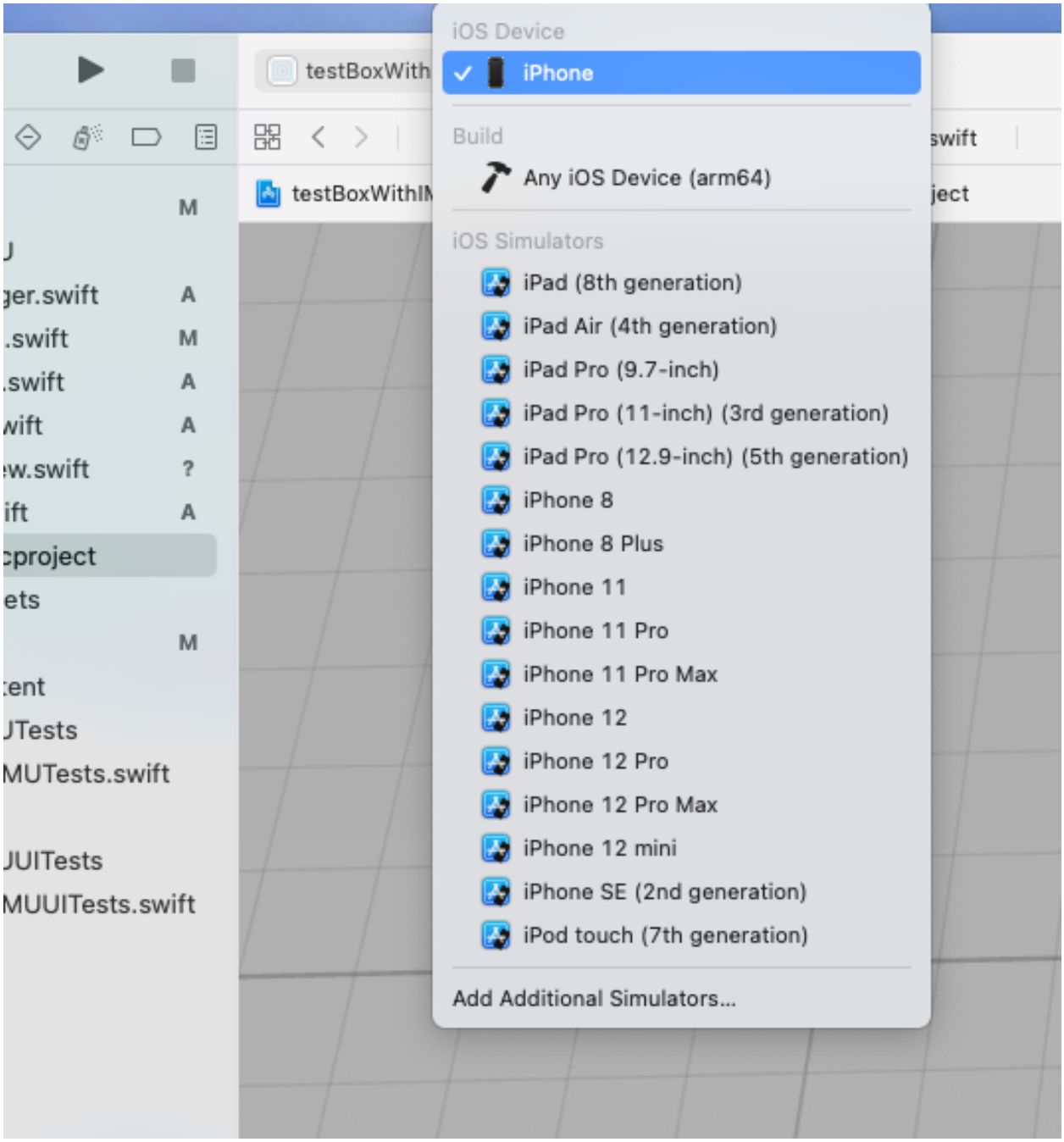


Figure 22: Select phone target to deploy to

Once deployed to your phone, you need to go to Settings:General:Device Management on your iPhone or IPad to authorize the computer and apps that you are developing, then when your app runs for the first time, it will ask to use Bluetooth Low Energy, so click “ok”. Also the app will ask to use your camera. The end result is an app that mirrors your accelerations and orientations. The app as written reads in a Quaternion for orientation taken from the BNO085 sensor, and three axes of acceleration. Accelerating causes the virtual AR device to bump in the direction of acceleration, while the quaternion is used to orient the device. Note that the coordinate systems need to be properly aligned between the BNO device and the app. Also I would suggest turning on the IMUBox level on the table in the same direction as the app starts, to keep things aligned. Here’s some more info on Quaternions: I added to the DataModel Published Variables for Quaternions that feed the rotation of the object.

```
    @Published var Qx: Float = 0 {
        didSet {translateRotateBox()}
       }
    @Published var Qy: Float = 0 {
        didSet {translateRotateBox()}
       }
    @Published var Qz: Float = 0 {
        didSet {translateRotateBox()}
       }
    @Published var Real: Float = 0 {
        didSet {translateRotateBox()}
       }

    init()  {
        //initialize ARView
        arView = ARView(frame: .zero)

        let boxAnchor = try! Experience.loadBox()
        arView.scene.anchors.append(boxAnchor)
    }


    func translateRotateBox() {
        if let imuBox = (arView.scene.anchors[0] as?
            Experience.Box)?.imuCase  {
            let xTranslationM = xTranslation/100
            let yTranslationM = yTranslation/100
            let zTranslationM = zTranslation/100

            let translation = SIMD3<Float>(xTranslationM, yTranslationM, zTranslationM)
            let rotation = simd_quaternion(Qx,Qy,Qz,Real)


            imuBox.transform.translation=translation
            imuBox.transform.rotation=rotation
        }
    }
}
```

Figure 23: Apply Quaternions to Rotations

https://medium.com/@jacob.waechter/scenekit-rotations-with-quaternions-d74dc6ba68c6

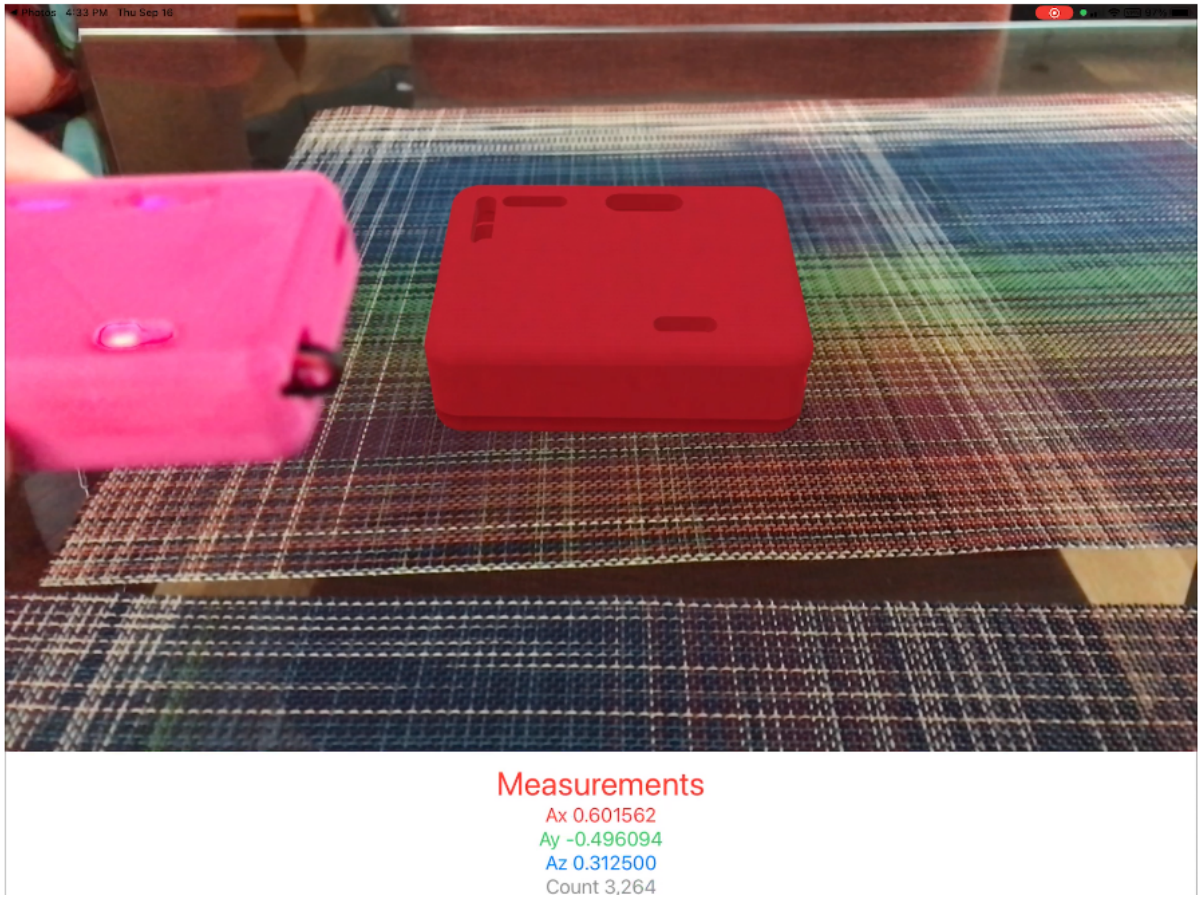


Figure 24: Enjoy!