2016/09/12

新规则发布以后，程序组结合新赛季的任务清单，思考自动程序方案，分组讨论

2016/09/14

成员提交得分策略以及实施方案

方案大概分为两种思路

停靠&按灯

方案一（停靠）：

[1 将机器人偏离竖直位置放置

[2 使用电机中的码盘值进行直走至角落漩涡边缘

[3 左右两侧的电机进行差速转弯

[4 使用电机中的码盘值进行直走至角落漩涡上

[5 停靠角落漩涡

方案二（撞球＋停靠）：

[1 使用电机中的码盘值进行直走至中心漩涡

[2 将本方大球撞离原始位置

[3 停靠中心漩涡

方案三（按灯）：

[1 将机器人偏离竖直位置放置

[2 开启 ods 传感器检测白线校准车体位置

[3 开启颜色传感器检测指示灯的颜色

[4 使用舵机的位置来进行按灯动作

方案四（按灯＋撞球＋停靠）：

[1 使用电机中的码盘值进行直走至中心漩涡

[2 左右两侧的电机进行差速直角转弯

[3 用时间控制进行直走

[4 开启 ods 传感器检测白线校准车体位置

[5 开启颜色传感器检测指示灯的颜色

[6 使用舵机的位置来进行按灯动作

[7 时间控制车体后退至中心漩涡

[8 将本方大球撞离原始位置

[9 停靠中心漩涡

方案五（按灯＋撞球＋停靠）：

[1 将机器人偏离竖直位置放置

[2 开启 ods 传感器检测白线校准车体位置

[3 开启颜色传感器检测指示灯的颜色

[4 使用舵机的位置来进行按灯动作

[5 时间控制车体后退至中心漩涡

[6 将本方大球撞离原始位置

[7 停靠中心漩涡

方案六（按灯＋按灯）：

[1 将机器人偏离竖直位置放置

[2 开启 ods 传感器检测白线校准车体位置

[3 开启颜色传感器检测指示灯的颜色

[4 使用舵机的位置来进行按灯动作

[5 通过电机控制麦轮使车体直接进行水平移动至检测到白线

[6 开启颜色传感器检测指示灯的颜色

[7 使用舵机的位置来进行按灯动作

方案七（按灯＋按灯＋撞球＋停靠）：

[1 将机器人偏离竖直位置放置

[2 开启 ods 传感器检测白线校准车体位置

[3 开启颜色传感器检测指示灯的颜色

[4 使用舵机的位置来进行按灯动作

[5 通过电机控制麦轮使车体直接进行水平移动至检测到白线

[6 开启颜色传感器检测指示灯的颜色

[7 使用舵机的位置来进行按灯动作

[8 两侧电机进行同向不同速动作

[9 时间控制车体后退至中心漩涡

[10 将本方大球撞离原始位置

[11 停靠中心漩涡

2016/09/19

冗长的中秋假期过后，团队恢复了紧张而又有序的训练

经过多方的协商，已经初步敲定了方案五来作为自动程序的基本框架

方案五（按灯＋撞球＋停靠）

[1 将机器人偏离竖直位置放置

[2 开启 ods 传感器检测白线校准车体位置

[3 开启颜色传感器检测指示灯的颜色

[4 使用舵机的位置来进行按灯动作

[5 时间控制车体后退至中心漩涡

[6 将本方大球撞离原始位置

[7 停靠中心漩涡

2016/09/21

由于在此之前对于自动程序没有过很深的研究，程序的编写工作开展的不是十分的尽如人意

于是，准备分模块进行自动程序阶段的编程任务，最后再将其汇总

2016/09/22

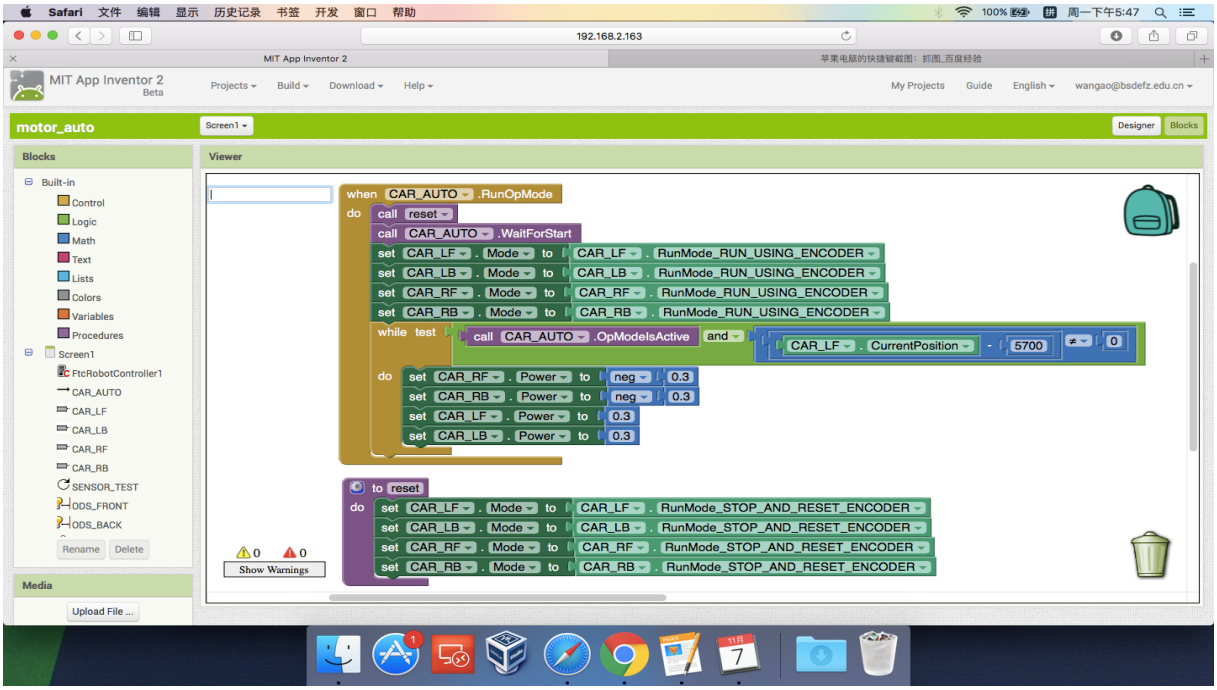
电机（**motor**）

encoder 的使用

tips：1、 使用之前需要 **reset encoder**；

2、通过给 **power** 赋值使电机转动从而才能使 **encoder** 的值有变化；

3、电机转一圈的值约为 **3000**；



4、while 判断语句中加入 **opmodeisactive** 才能在运行过程中通过驾驶员端进行 **stop** 等操作；

5、过程中可以 **reset**

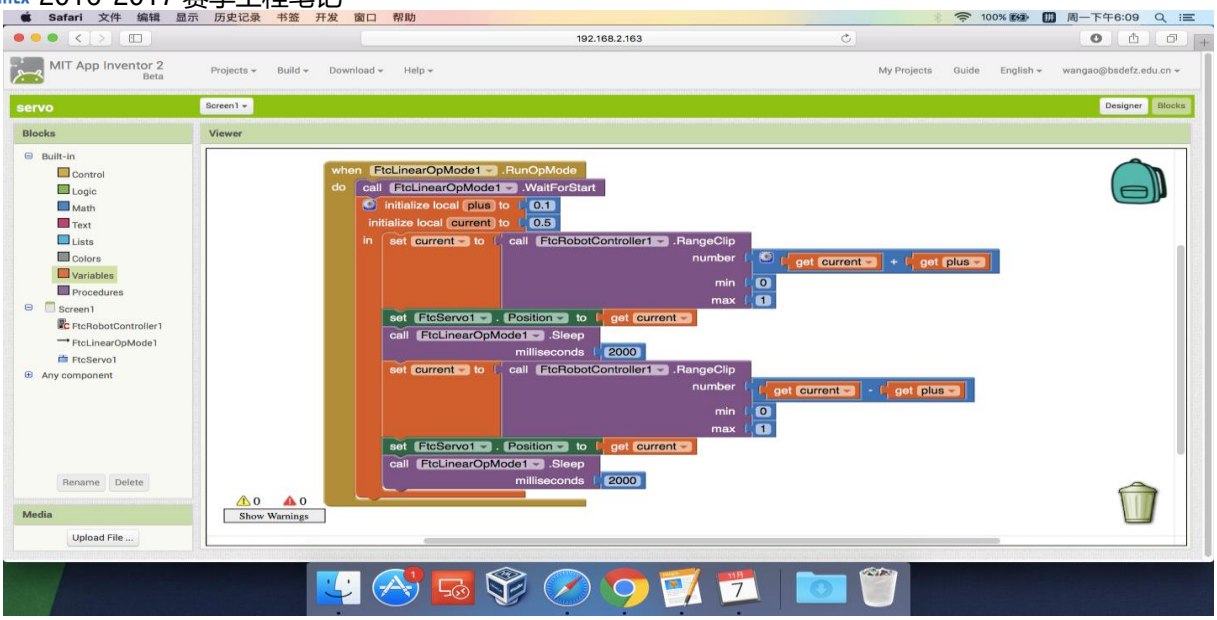
2016/09/23

舵机**(servo)**

position 的使用

tips：1、通过简单的算法将 **servo** 的目标位置表达出来；

2、注意需要控制最值避免损坏舵机；



3、**sleep** 语句是指运行上一条指令一段时间后停止并进行下一条

2016/09/27

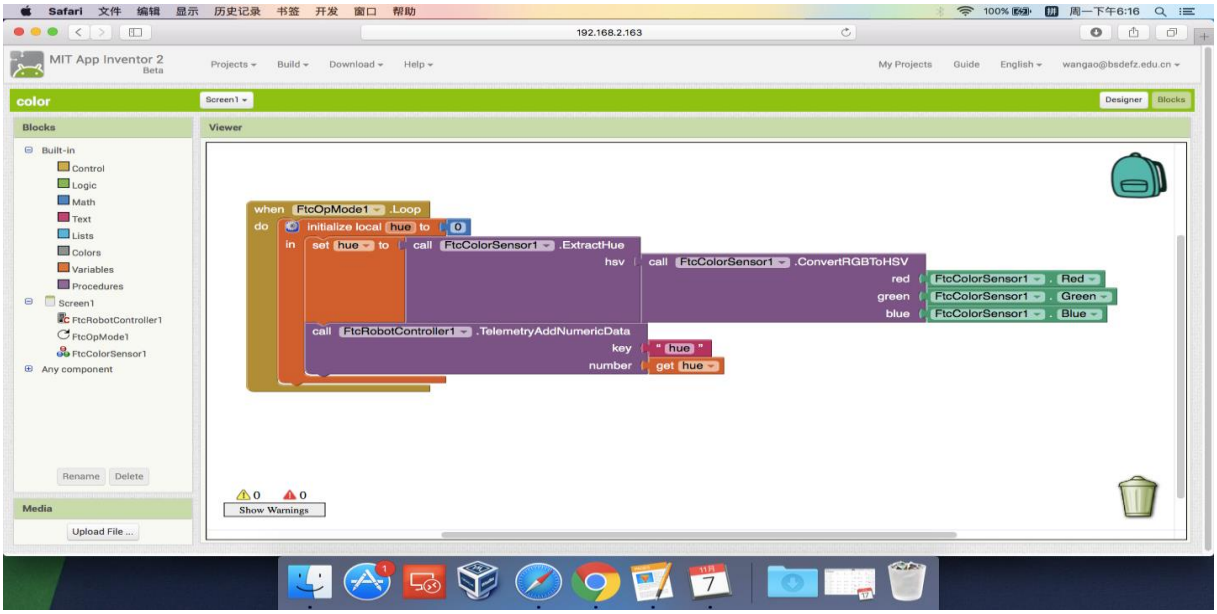
颜色传感器（**color sensor**）

hsv 与 **rgb** 之间的转换

tips：1、仅使用 **hue** 值就可以直接分辨红色与蓝色

2、设置变量时要注意是局部还是全部

3、传感器返回数值时最好用 **text** 为属性

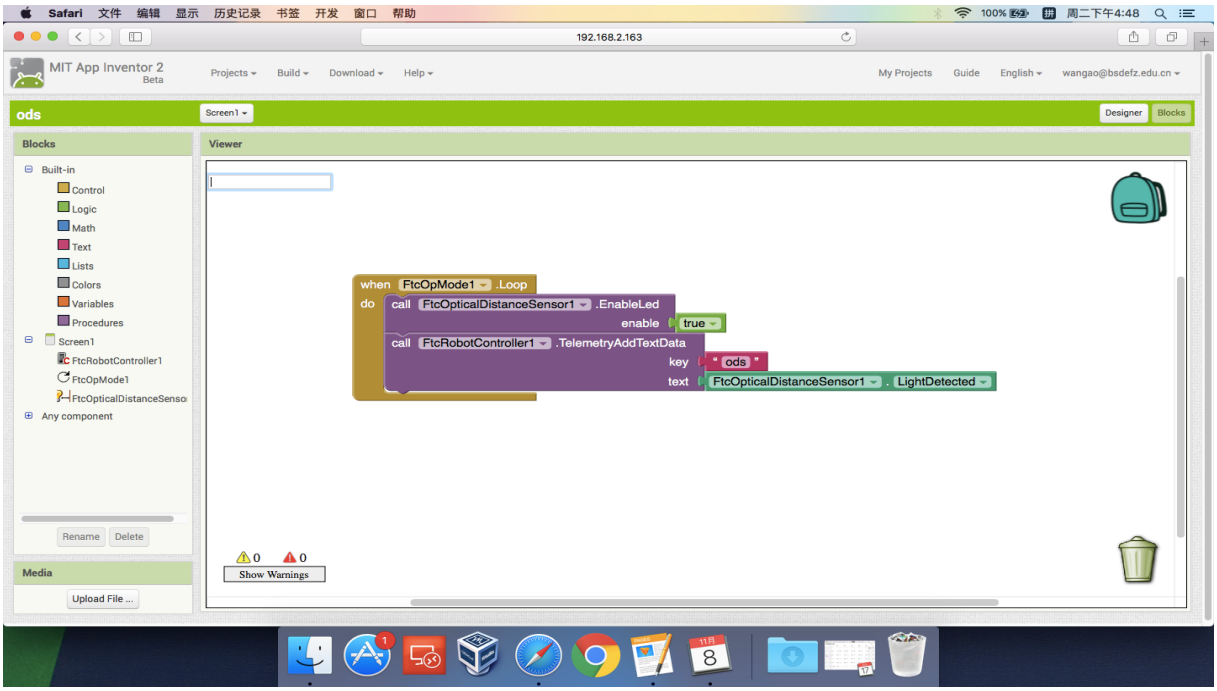


4、检测数值时因为需要传感器循环检测当前位置的 **hue** 值所以程序需使用 **opmode** 而不是 **linearopmode**

2016/10/10

光学距离传感器**(optical distance sensor)**

检测黑与白



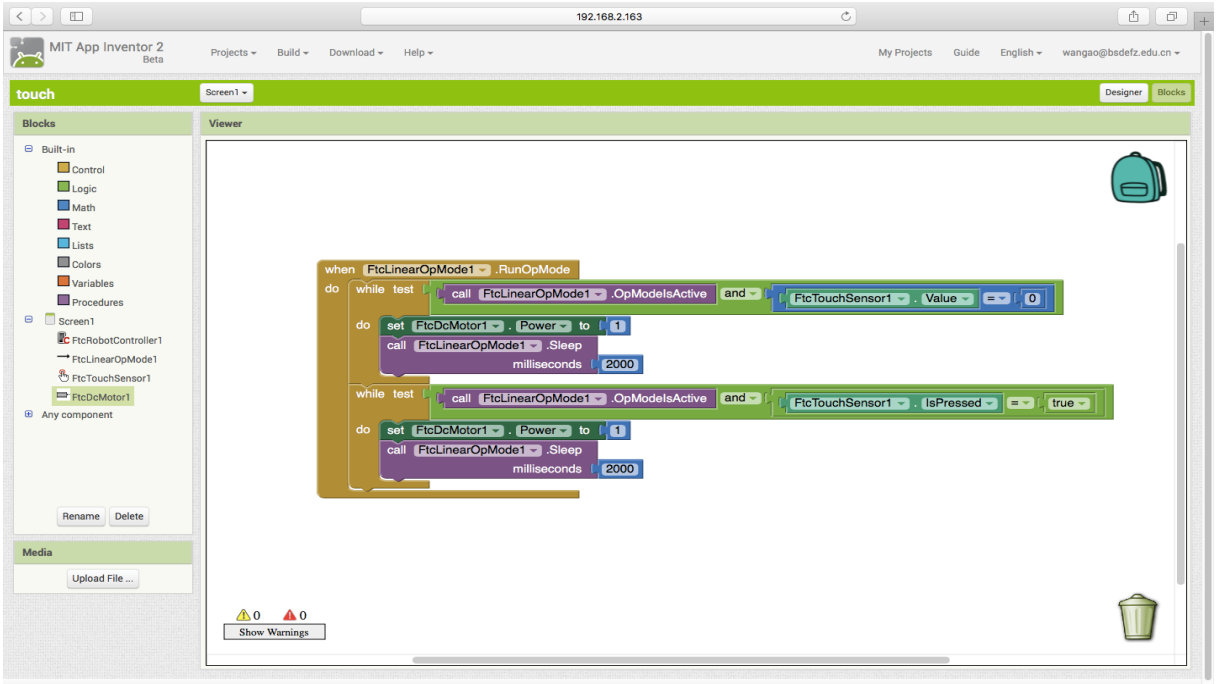
tips：1、使用之前需要先判断 **enable** 为 **true**

2、传感器返回数值时最好用 **text** 为属性

3、黑色数值偏小

2016/10/12

触碰传感器（**touch sensor**）



按下&滞空

tips： **1、图为两种方法控制触碰传感器**

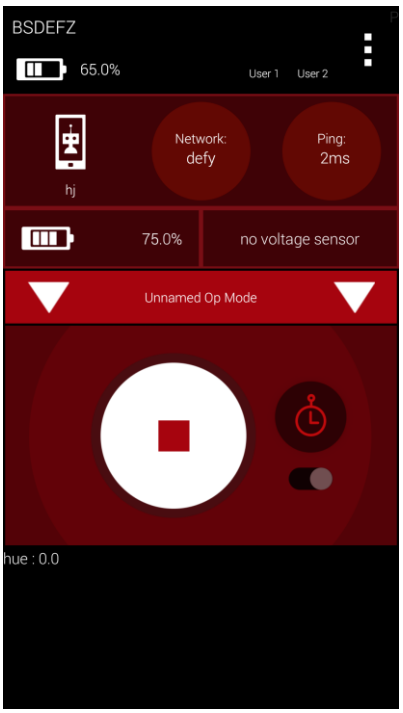
2、判断 touch sensor 的 value（0&1）

3、判断 touch sensor 的 ispressed 状态（true&false）

4、判断 true&false 需要使用逻辑等于

2016/10/14

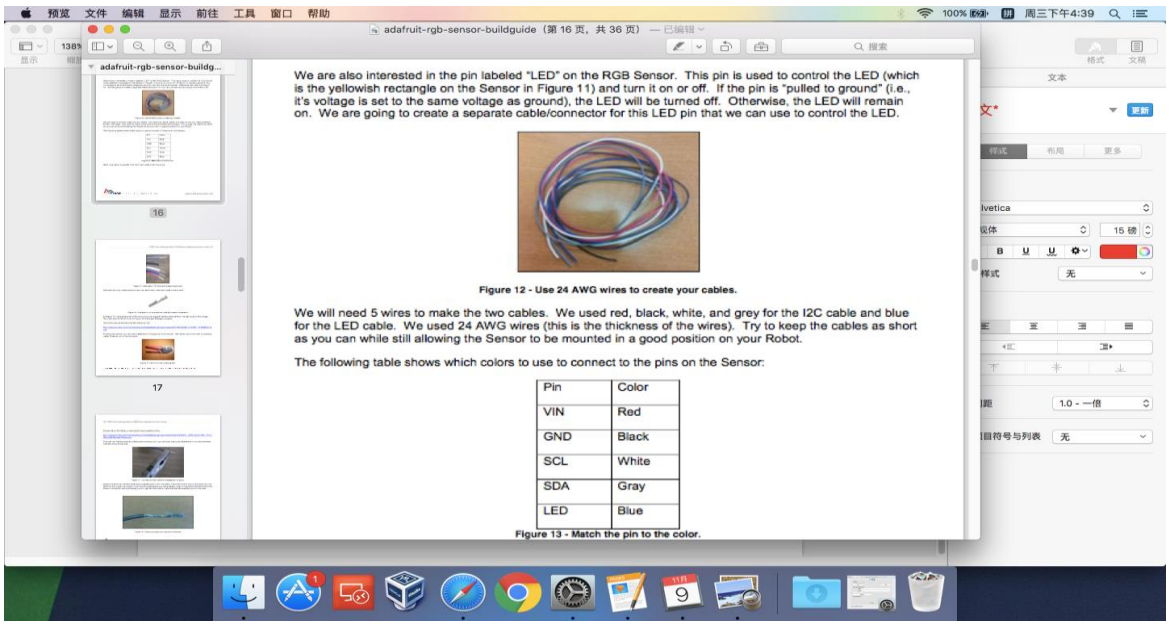
虽然各模块程序的编写已经基本成型，但是在颜色传感器的使用方面还是遇到了些困难，使传感器无法成功返回 **hue** 的数值（如图）



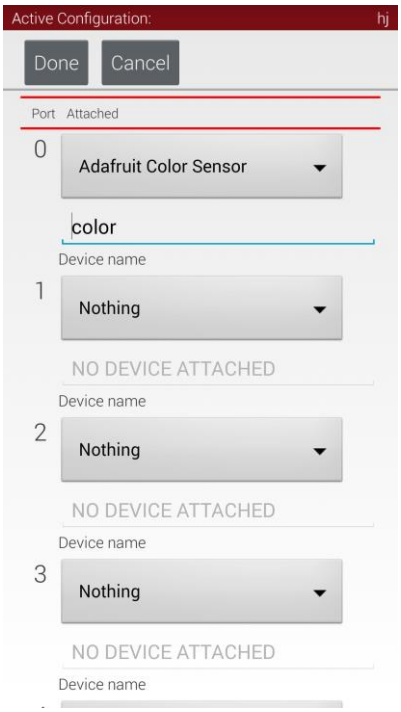
2016/10/17

在查阅过相关资料后，终于发现了问题

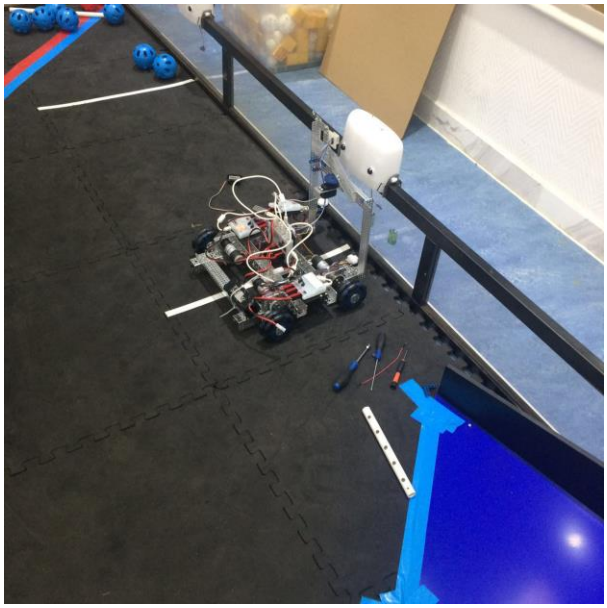
1、由于颜色传感器不是官方配件，所以颜色传感器的组装需要自己动手。将电路板与导线相焊接并 **3d** 打印出相应保护套。根据安装手册发现，有两根导线插接错误



2、在配置文件时，端口的属性选择出了些问题,在颜色传感器的使用时，我们应该选用端口的属性时 **adafruit color sensor** 而不是 **color sensor**。



2016/10/19

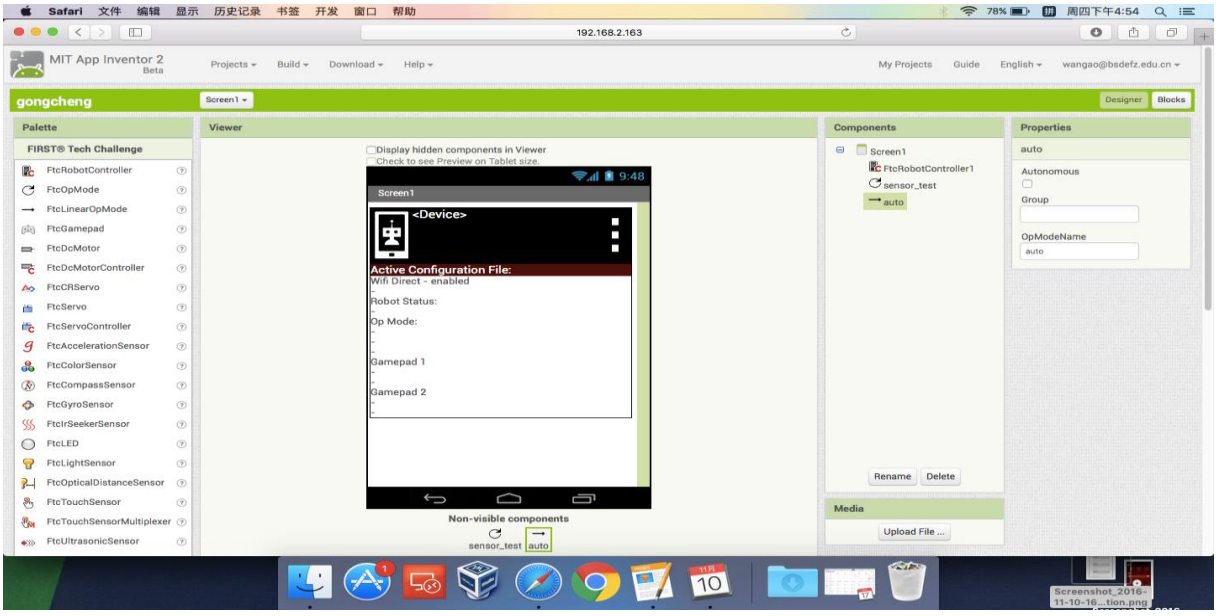


由于自动程序的测试与搭建车体时时间有些冲突，两者无法同时进行。于是编程组的成员们搭建了一个底盘车并且将所需硬件与传感器等都安装在上面，以此来进行专门的自动程序检测。

2016/10/21

所有的基本硬件与软件模块已经准备就绪，于是接下来我们就进入了硬件与软件之间的配合测试。

为了不重复下载 **apk** 来进行传感器测试与自动程序之间的转换浪费时间我们在一个 **project** 里面同时建立了两个 **mode**，一个为 **op mode** 来进



行颜色的数值检测，另一个为 **linear op mode** 用来执行自动程序。

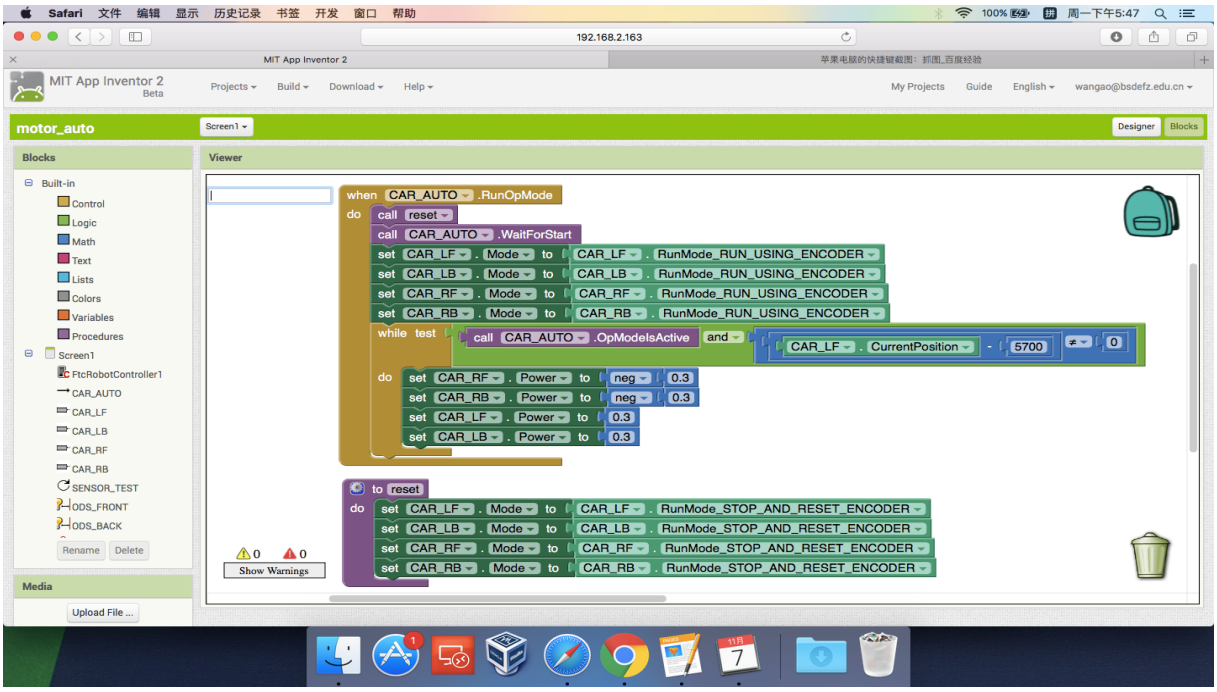
2016/10/24

使用码盘的值来控制车进行直走然而并不成功。

问题：

1、不能直接给码盘进行赋值，只能通过实时对码盘值进行检测才能控制

2、判断语句需要使用 **while test** 而不能是 **if**。在 **linear op mode** 中程序只会按照编写顺序从上至下执行命令，如果使用 **if** 判断，程序只能检测一次而不会按照我们的想法循环检测。



3、**while test** 语句会循环检测传感器（**color sensor**、**touch sensor**、**optical distance sensor**、**motor encoder**）数值，如果以达到目标值，**while** 判断为假，则程序会自动跳出 **while** 循环，向下进行。

2016/10/26

解决了直走的问题后，机器人能够走到指定目标位置，下一步是转弯。同样是利用码盘的值来进行相应动作，但还是不成功...

问题：**1**、电机转一圈的值大概为 **3000**，我们在设置目标值的时候，数值过小，因为机器人本身有一定的质量，所以会导致电机动作不明显，在宏观上表现为机器人没有任何动作，一段时间后，直接结束程序的运行

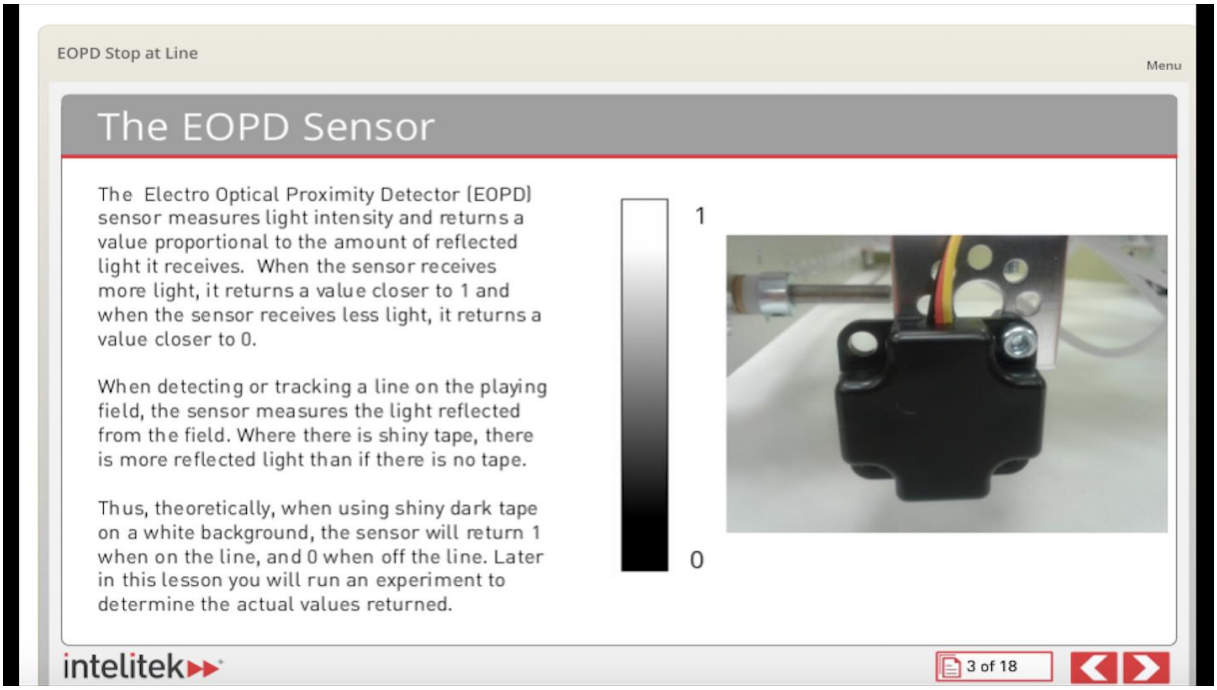
2、不太明白 **stop_and_reset** 语句中 **stop** 的作用

建议：**1**、如果前面有直走的程序，建议在转弯之前可以 **reset** 码盘值，这样会简单一点

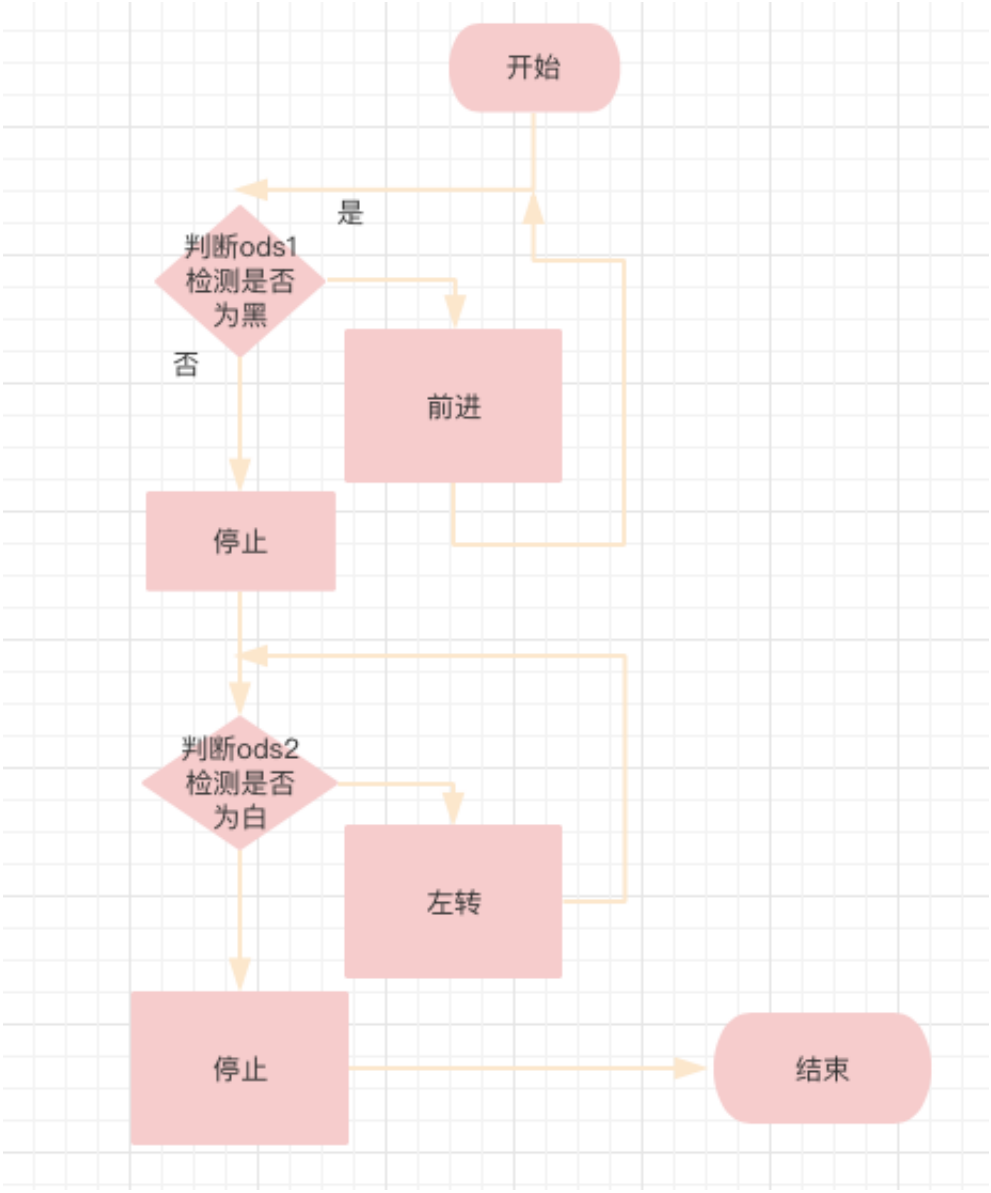
2016/10/27

使用码盘值进行底盘操作时，因为编程软件的原因，虽然程序能够读取各个电机的码盘值但是我们无法通过程序去同时控制四个电机的码盘值，而且由于电机都不是很新，其内置的码盘值计数传感器都有一定程度的磨损，导致各个电机如果纯靠码盘值来控制无法完全同步，所以我们就战略性的放弃了纯通过码盘值去控制机器人到达指定位置。

后来，我们选择了用 **optical distance sensor** 去检测白线的方法使机器人行驶到指定目标位置，这样不仅会减少编程的难度，而且会使机器人更



稳，到达指定目标位置的准确率更高，只不过判断逻辑会稍稍有些费劲且机器人的起始位置的摆放要求也会比较的严格。



2016/10/28

经过小组之间的讨论，程序的总体框架流程图已经大致写出来

2016/11/07

第二版自动程序已经写好了。但是经过一轮又一轮的测试，一些问题也随之暴露出来。首先是颜色传感器对灯的颜色识别能力不是太好，无论颜色检测到是什么，它都会传回蓝色的数值，从而导致舵机只往一个方向转。另外，按灯的方式也需要进一步的改善，就算是单纯去按灯，也无法准确的碰到指示灯中央的按钮，然而接下来关于按灯装置的硬件结构就是搭建组的事情了哈哈□

2016/11/09

成车已经基本成形，剩下的就是一些加固与收尾的工作，那么现在就轮到我们去写手动程序去操控机器人。手动程序相比于自动程序来说，稍稍略显简单，啊不，是太太太简单了。

反正也闲着没事干，那就把手动程序的编写方法大概的写写吧。

2016/11/10

首先呢，在页面的正中央有一个类似于手机屏幕的显示窗口，没错，这就是安装在机器人端，与主控制器相连接的手机。在页面的左侧有一列叫做 **palette** 的任务栏，它将所有我们可能在 **ftc** 比赛中用到的可以进行操控的硬件罗列了出来，而我们需要做的就是，把我们真正需要用的模块拖到手机屏幕上。

我们以通过一个手柄的两个摇杆来控制机器人行走以及另一个手柄的按键值来控制舵机的角度为例来具体的进行编程说明。

1、将 **ftc robot controller**、**ftc op mode**、**ftc gamepad(x2)**、**ftc dcmotor(x4)**、**ftc servo** 拖入手机界面（注意：为了完成通过手柄来控制车体的任务，程序需要无限的循环检测手柄的各部分值，所以在模式选择中，我们应该在手动程序中选择 **op mode**）

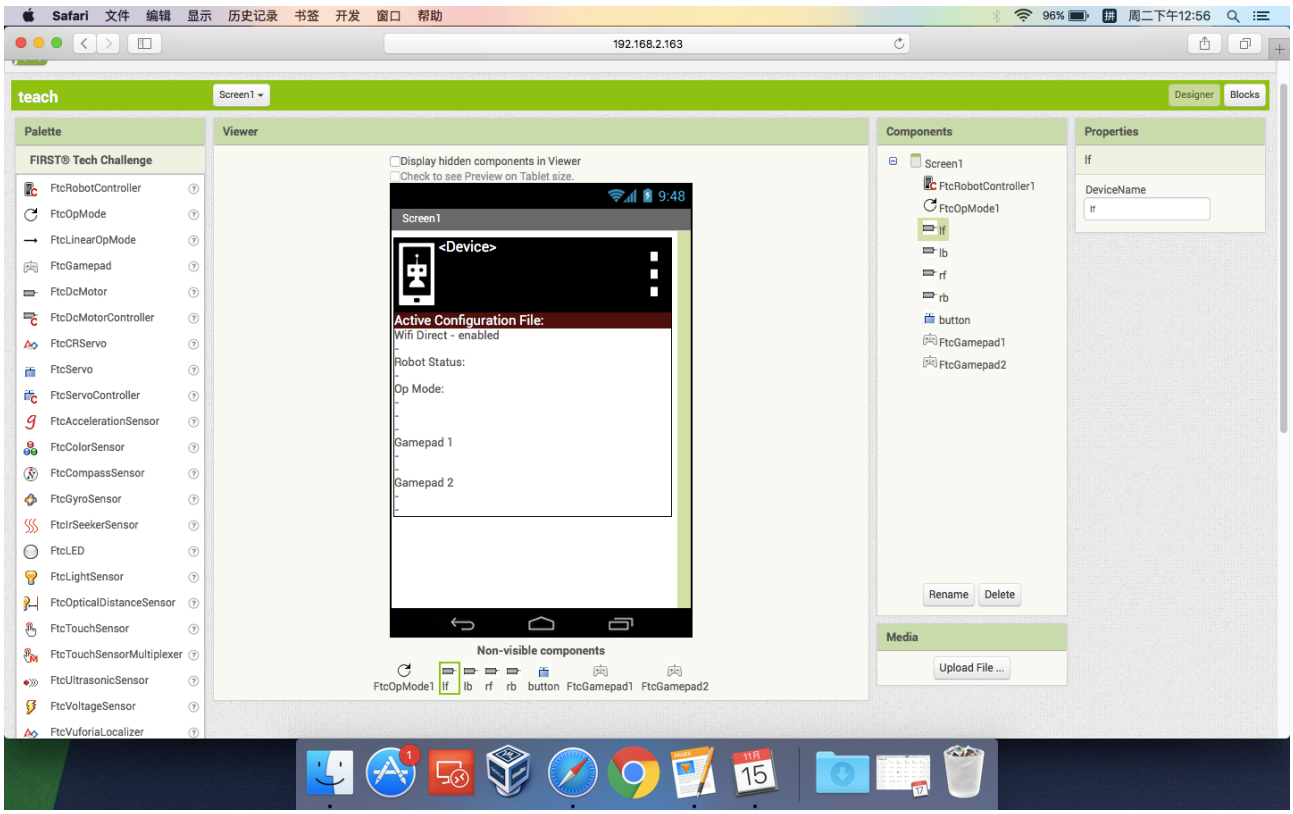
2、将各个执行模块的 **device name** 改成与相应任务相关的名字，同时 **rename** 一下（注意：**device name** 一定要记住，因为在与手机之间进行配置的时候，需要在各个端口输入其相应的 **device name**）

2016/11/11

首先祝大家光棍节快乐（以吾之名，召唤 **fff** 团，烧死异性恋！！）

1、点击右上角 **blocks** 进入编程界面

2、首先我们需要将左侧（右侧）前后的两个电机的方向全都反过来（注



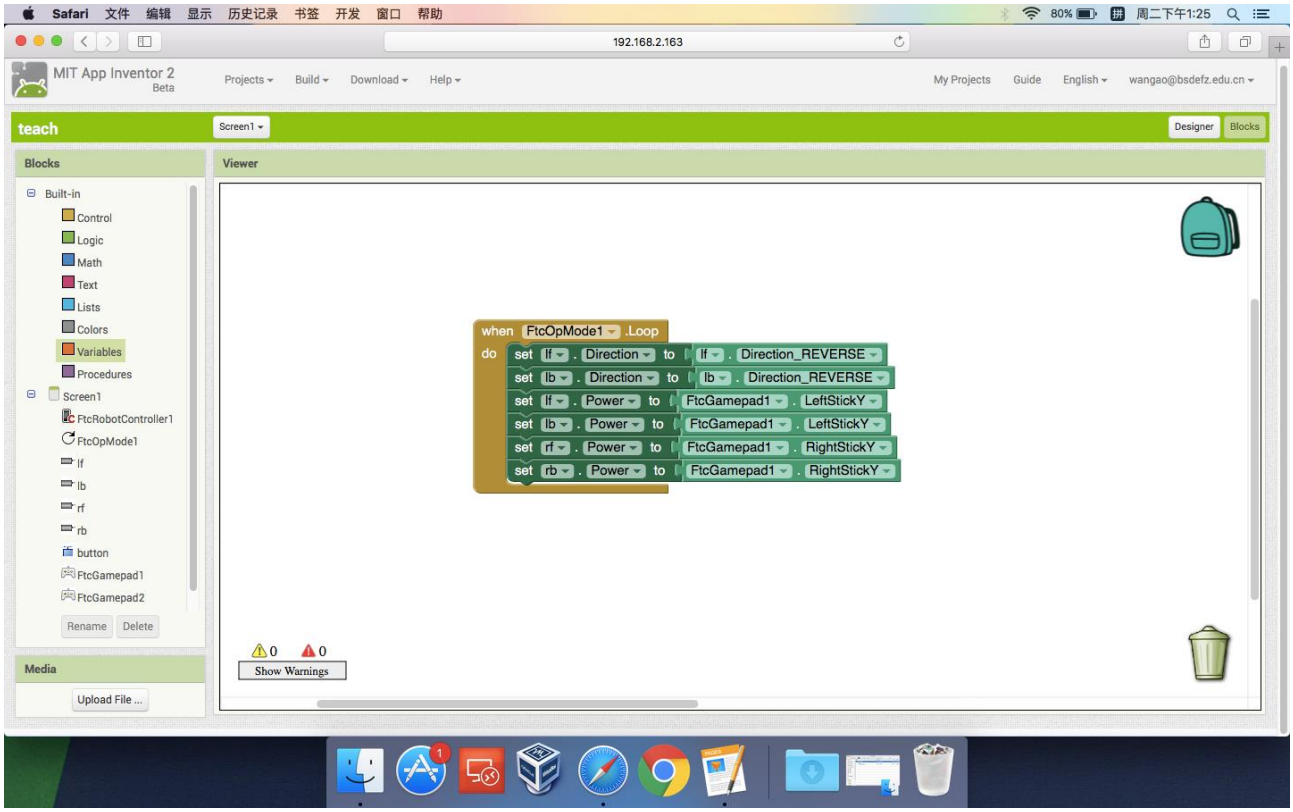
意：所有的电机都是一样的，其都有一个内置的传感器，逆时针转为 **1**，顺时针转为－**1**，一般的车都是将两侧电机相向而装，所以为了方便程序猿的编程工作，所以我们需要在起始位置将任意一侧的前后两个电机的方向调成相反的）

3、为了方便驾驶员的操作，我们决定采用坦克式的驾驶方式来进行程序的编写，根据资料可得知，手柄的两个摇杆其内部均为一个平面直角坐标系，纵向为 **y** 轴，横向为 **x** 轴，摇杆所处位置不同时，即数值也不同，于是我们利用摇杆这一属性，去编写电机的程序。给左侧电机赋予左摇杆的

值，给右侧电机赋予右摇杆的值。（如果我们已经使用了左右两支摇杆的 **y** 轴去控制车的底盘，那我们接下来就应该尽量避免再去给两支摇杆的 **x** 轴进行赋值任务，因为摇杆很灵活，在驾驶员操控时，很容易误触）

2016/11/14

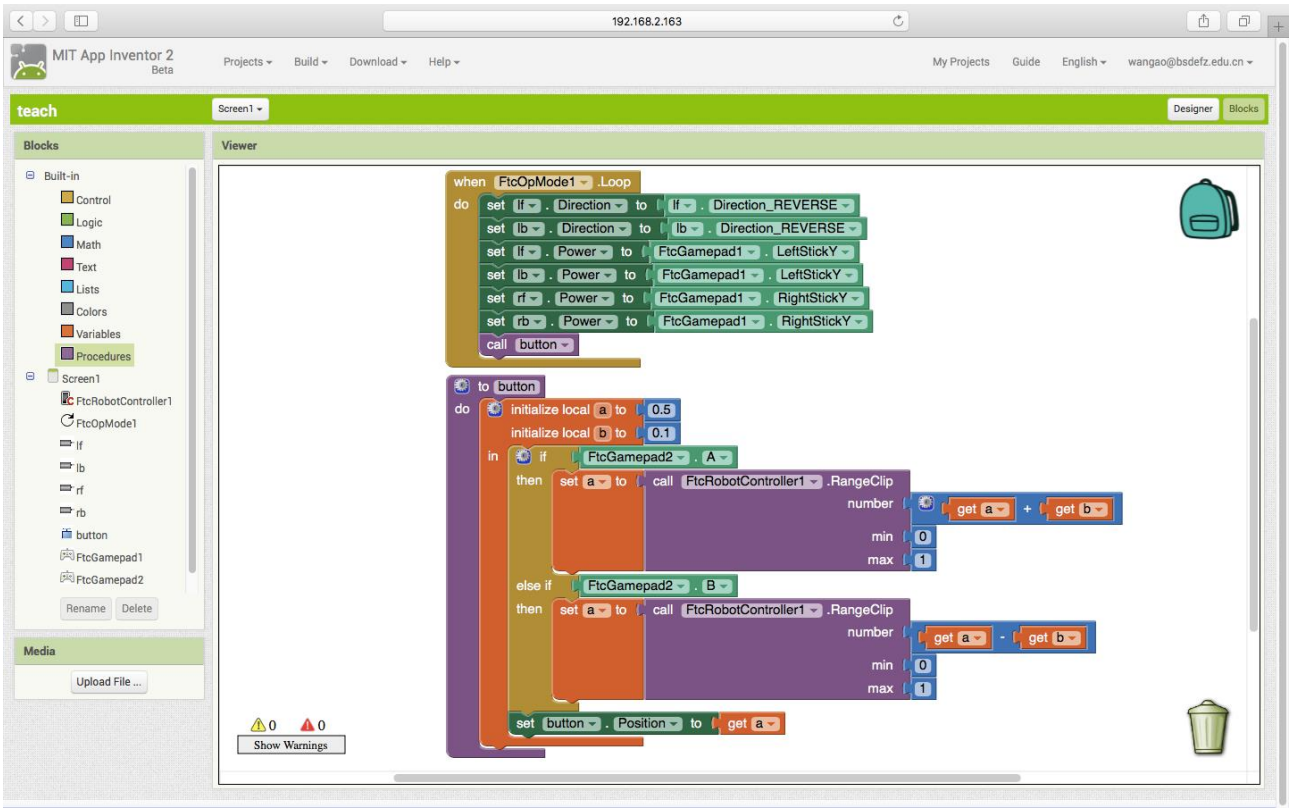
1、在舵机程序的编写中，由于指令数目过多，我们可以定义一个子程序来进行编写（在主程序中我们需要进行调用，才能使子程序有效）



2、我们采用舵机的程序逻辑与电机的稍有不同，首先需要定义两个变量 **a**、**b**（变量分为局部变量与全局变量，在此，我们选用的是局部变量）**a** 为当前值，**b** 为改变值

3、如果按下 **A** 键，则当前值 **a** 被赋值为 **a+b**，如果按下 **B** 键，则当前值 **a** 被赋值为 **a**－**b**，最后舵机的位置将会被赋值为 **a**。（在计算过程中，我们应有保护舵机的理念，该注意其最大值与最小值，避免损坏舵机）

2016/11/15



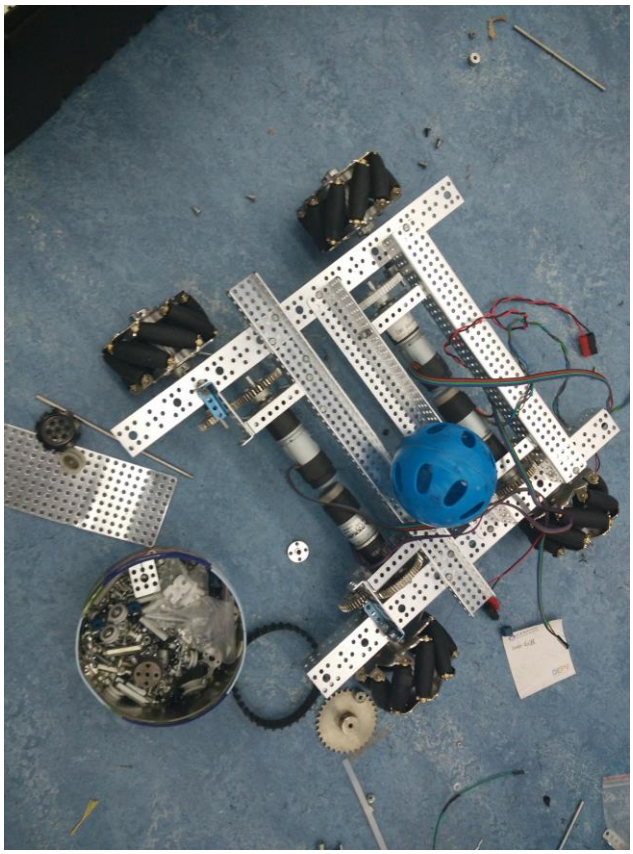
手动程序无非就是电机与舵机的控制（电机也可以通过定义变量进行赋值的方法进行控制）

在手动程序编写完成后，我们尝试了对麦轮控制的程序编写

首先，通过资料的查阅，我们了解到通过控制电机转向的逆时针与顺时针，车体可以进行前进、后退、向左平移、向右平移。如果我们能够成功控制其完成以上指令，这对于我们自动程序按灯的方案有很大的益处。于是我们先打了一个装有四个麦轮的底盘车。

2016/11/16

在研究其左右平易时电机方向规律后，随之我们编写了一套有关控制机器

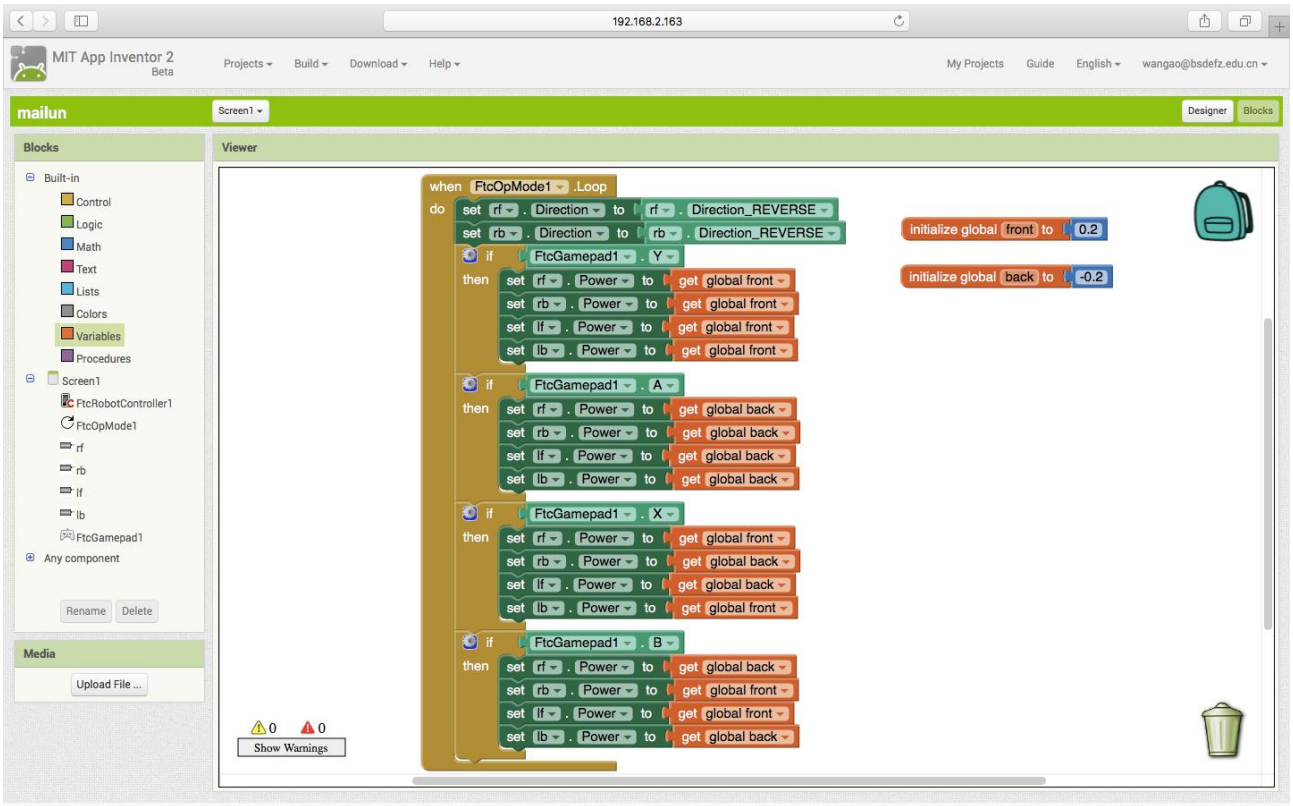


人行驶方向的程序，应用起来也很成功

通过研究，发现如果控制同侧两轮方向相反，异侧两轮方向相同即可使机器人进行水平方向的平移

1、这回我们使用的是进行全部变量的变量方法来编写

2、为了防止机器人在运行时，失去控制，我们将电机的 **power** 值赋的稍小。



2016/11/18

我今天学习了 windows 上如何进行 adb 操作以直接进行 apk 包的安装，尽管结果不尽人意，但是我还是学到了一些东西。

首先我尝试直接在命令行中调用 adb shell，然而系统显示系统中没有这个东西，后来我发现原来 adb 是一个软件，必须先要安装才可以调用。

于是乎，我上 baidu 进行搜索，找到许多个版本，果断下载最新版。下载后我按照安装说明放入系统变量所在目录中。

接着就可以进行调用了。在命令行中输入 adb shell 发现进入了 exe 的操作命令行，接着我无论是进行 adb install 还是 adb devices 出来的结果都是十分神奇。并没有成功运行的显示，手机上也没有新的软件出现。

我上网查找了 devices 为何没有各种各样的序列号出现，有的回答显示没有手机安装驱动，我重装驱动，有的回答显示是豌豆荚等手机助手占用了 adb 于是我又强行关闭了一系列相关进程。结果还是不行

最后，我退出 adb.exe 设置界面，再次查找 devices，终于，出现了设备好，并且可以正常地查看手机 cpu 等信息。

我尝试利用 adb install<>来进行安装，发现显示语法错误。

最后去除尖括号，终于安装成功。

经过这次事件我明白了我应该先详细了解一个功能的本质之后，才能惊醒操作。

要点提示：

- 百度下载 adb.exe
- 打开命令行工具
- 进入 adb.exe 所在目录（不知道怎么办的输入 help 按回车）

- 输入 adb devices 进行设备检查（有设备的话直接跳到 8）
- 没有查到的话重新连接手机，选择信任，打开 usb 调试
- 再次输入 adb devices 进行检查
- 输入 adb install 再拖入 apk
- 你要是还不会我也没辙了。

```
C:\Users\sd>adb install C:\Users\sd\Desktop\tengxunxinwen.apk
2859 KB/s (27444979 bytes in 9.371s)
        pkg: /data/local/tmp/tengxunxinwen.apk
Failure [INSTALL_CANCELED_BY_USER]

C:\Users\sd>adb install C:\Users\sd\Desktop\tengxunxinwen.apk
3411 KB/s (27444979 bytes in 7.855s)
        pkg: /data/local/tmp/tengxunxinwen.apk
Success
```

```
C:\Users\sd>adb devices
List of devices attached
e656be3c        device
```

如何使用 mac 编写 bash 脚本，并利用它来进行 apk 包的快速安装。

首先需要找到一个 apk 包，（mac 和 win 的不一样哈哈哈哈哈哈）。我先尝试了找到 AS 的自带 adb，然而并没有什么卵用。adb 包只是 win 版附带的。于是我又一次召唤了万能的度娘。找到并且下载了 adb 包。

尝试 bash 格式的过程是极其痛苦的，之前我都没有见过。一开始，各种无法调用这一个程序包，后来伟大帅气的张老师教给我格式应该是（./wenjianming）。正确地检查了 adb 包的版本号后进行了下一步尝试，因为 adb 包的内置原理是一样的，所以，机智地利用 adb devices 查看了当前设备信息。之后，我就进入 nano 进行脚本的编写，首先我利用 baidu 找到了开头格式#! / bin / bash。

接下来就是正式地编写了，我首先利用绝对路径进行 adb 是否可用的调试，事实证明，不行。

我和老师发现了各种编写的小错误，单词拼写，路径大小写。但是，最后把这些所有的错误都纠正后还是运行不了，于是上网寻找 bash 教学。最后发现应该利用 chmod 进行赋予权限才可以执行，不管 sudo 的事哈哈哈哈哈哈。

接下来，在原有代码段地下加入了 install 命令。