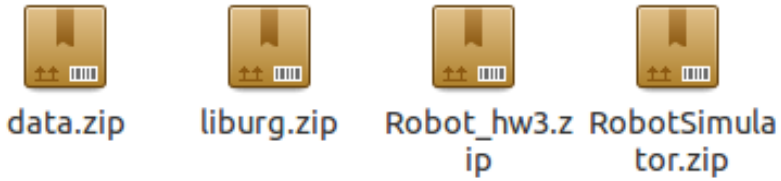
实习课3：使用Simulator进行本地数据回放

1.Files



Data.zip:数据包

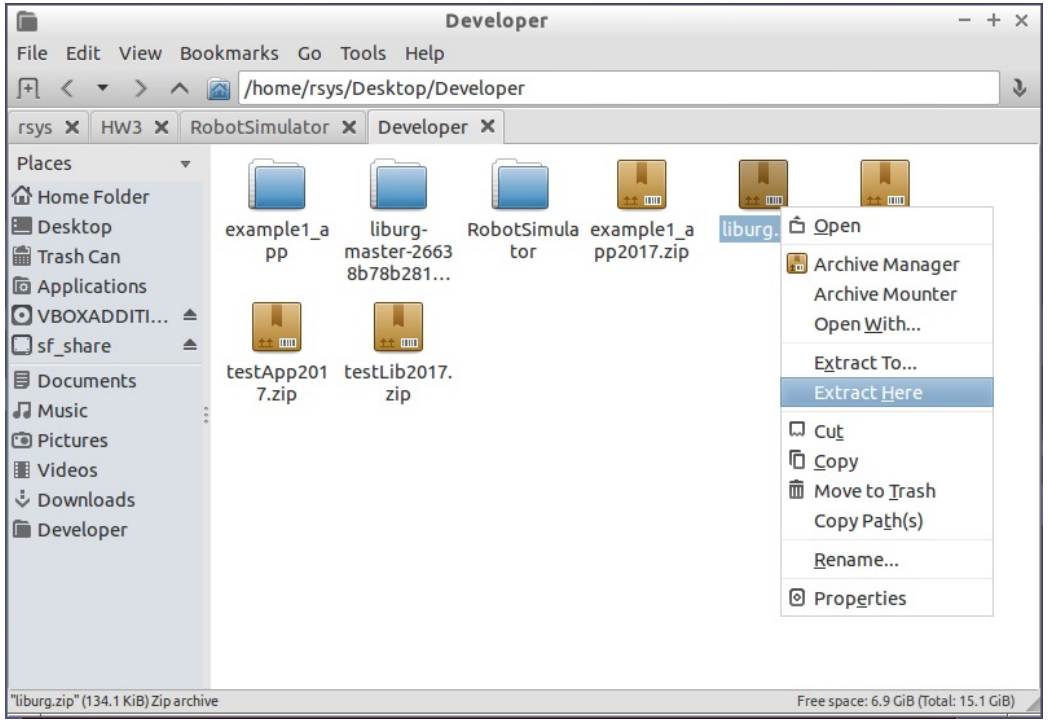
Liburg.zip:激光驱动

Robot.zip:底层程序

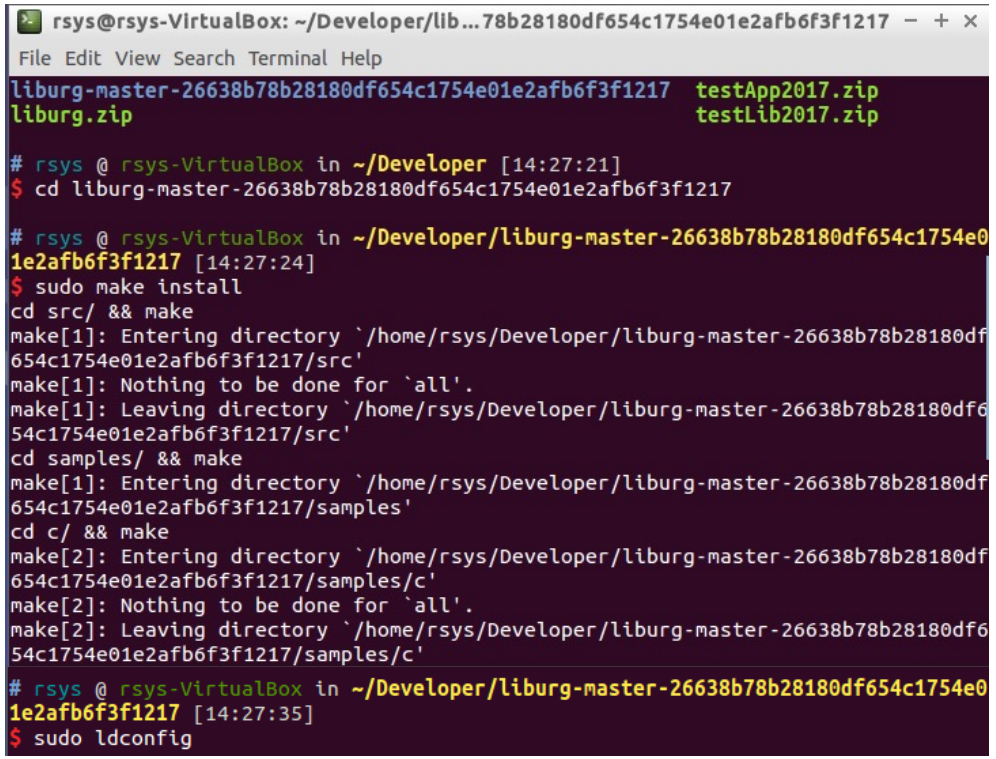
RobotSimulator.zip:高层程序

2. Liburg

#1:拷贝Liburg.zip压缩包至Developer文件夹
#2:直接/使用命令行　解压



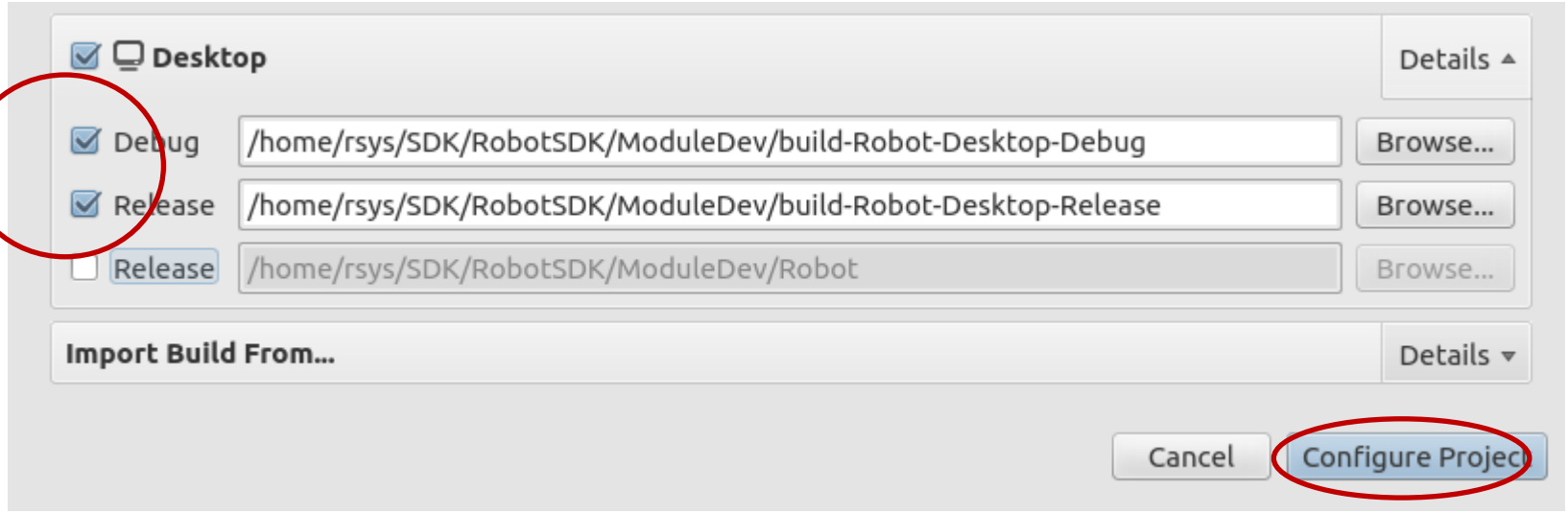
#3:命令行进入对应的文件夹liburg*
#4:　　sudo make install
　　sudo ldconfig



*Checkpoint 2

3. Dev&App

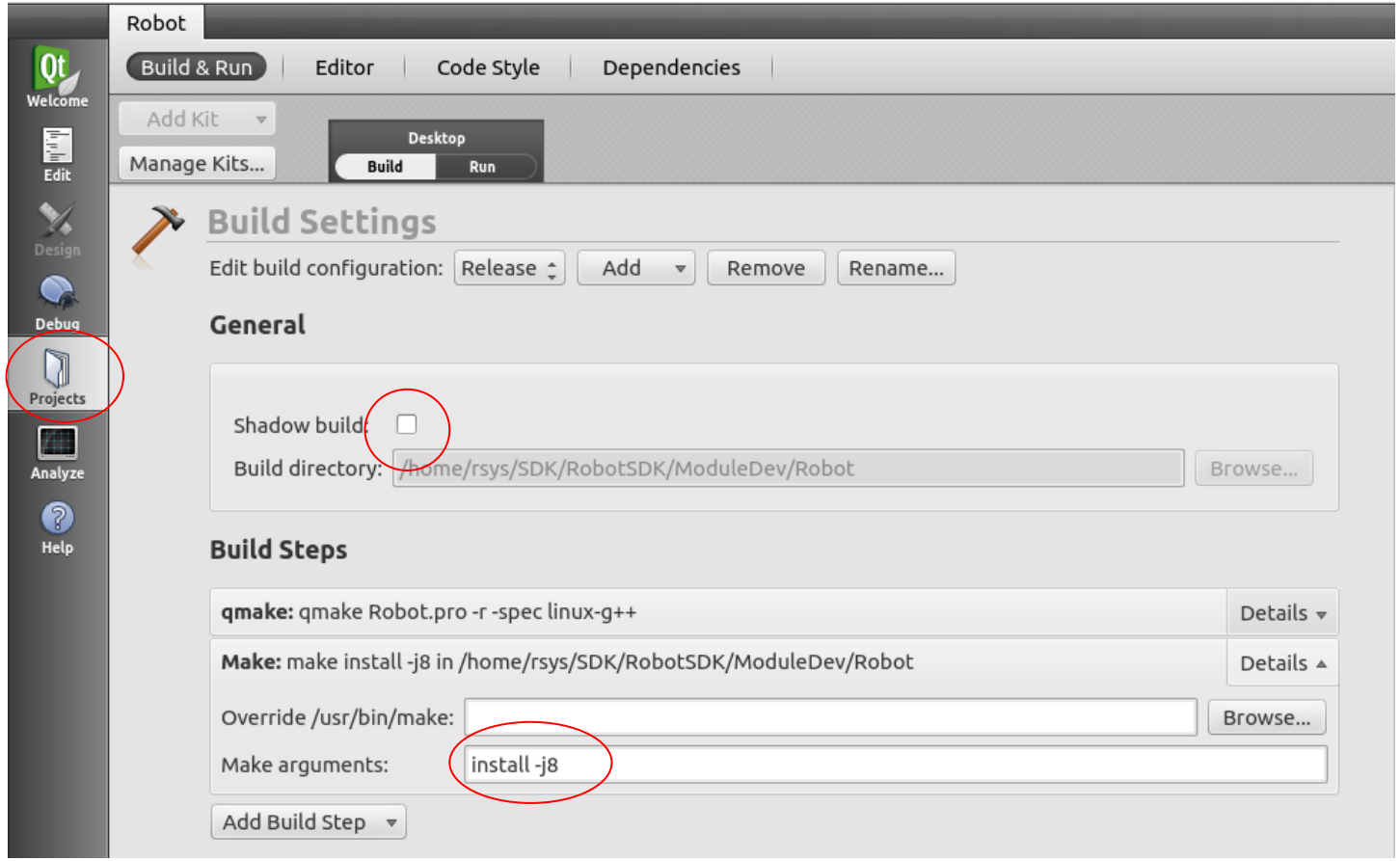
#1:底层程序拷贝至ModuleDev文件夹，解压得到Robot文件夹，进入打开Robot.pro
#2:检查配置

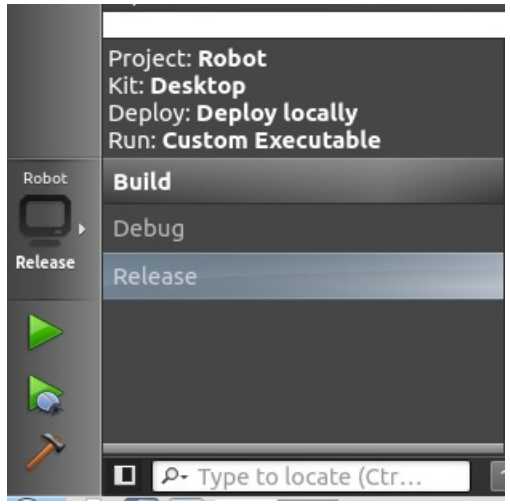


#2* 如果配置不对，可以删除Robot文件夹中的.user后缀文件，之后重新打开Robot.pro

3. Dev&App

#3:检查编译设置，没有问题后进行编译



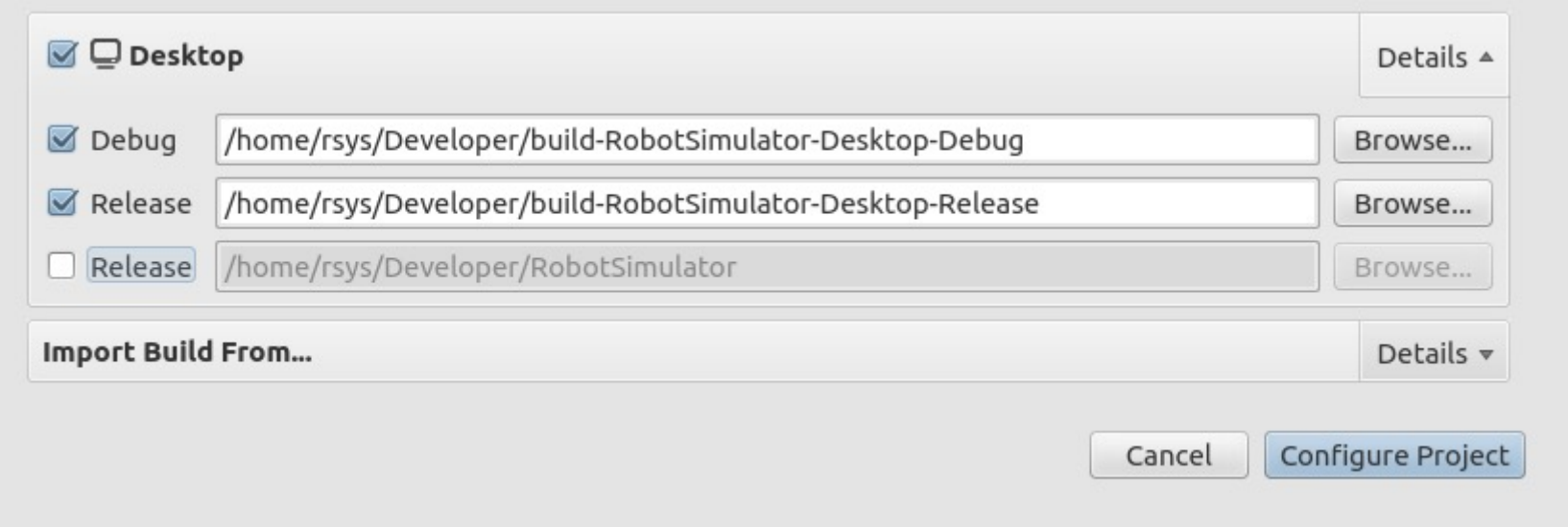


快速切换Debug/Release模式

*Checkpoint 3

3. Dev&App

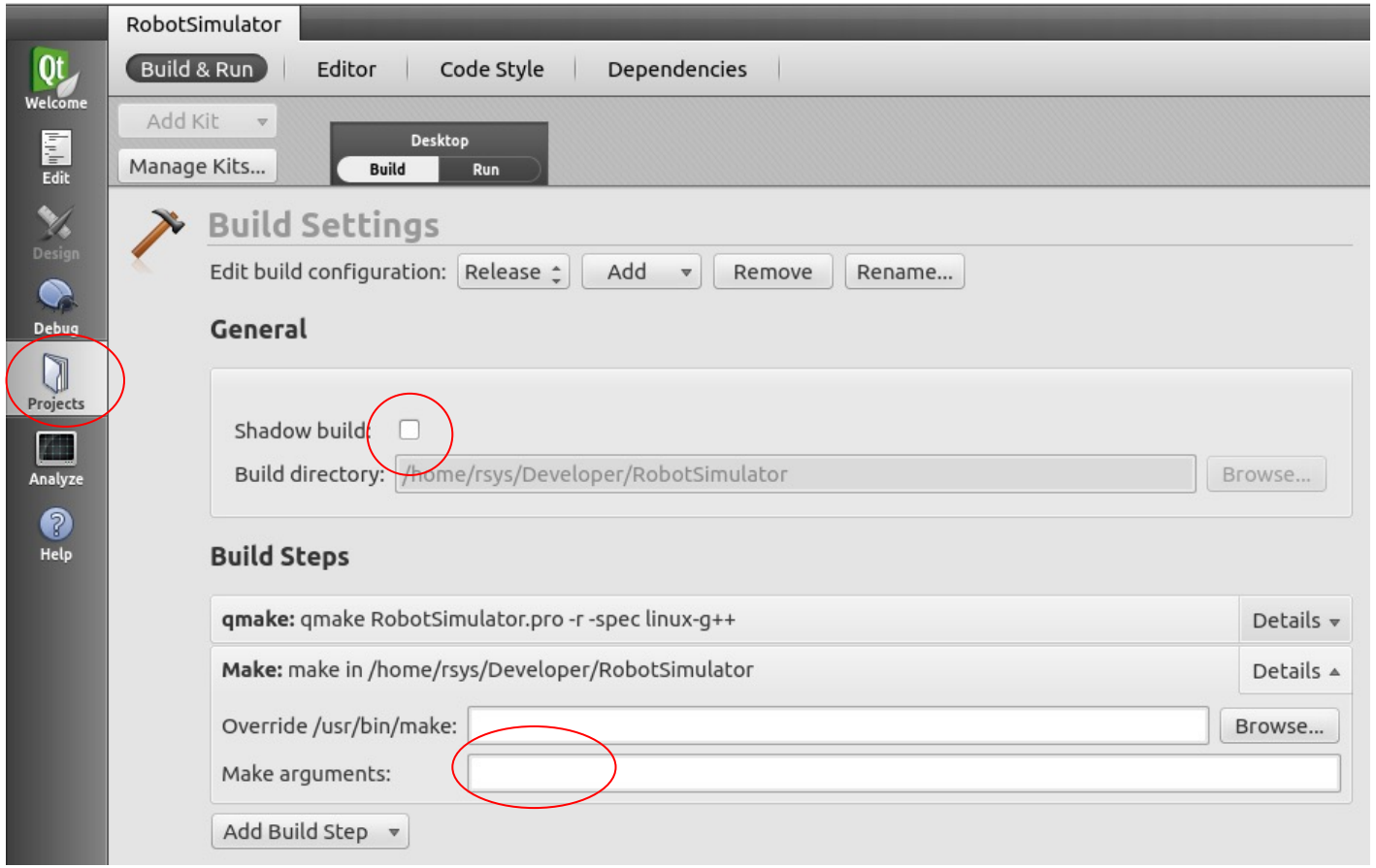
#4:高层程序拷贝至Developer文件夹，解压得到RobotSimulator文件夹，进入打开RobotSimulator.pro
#5:确认配置正确

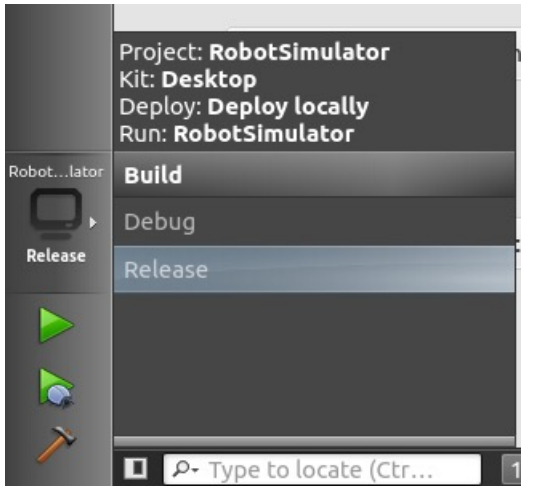


*如果配置不对，同样可以删除RobotSimulator文件夹中的.user后缀文件，之后重新打开.pro

3. Dev&App

#6:检查编译设置，没有问题后进行编译



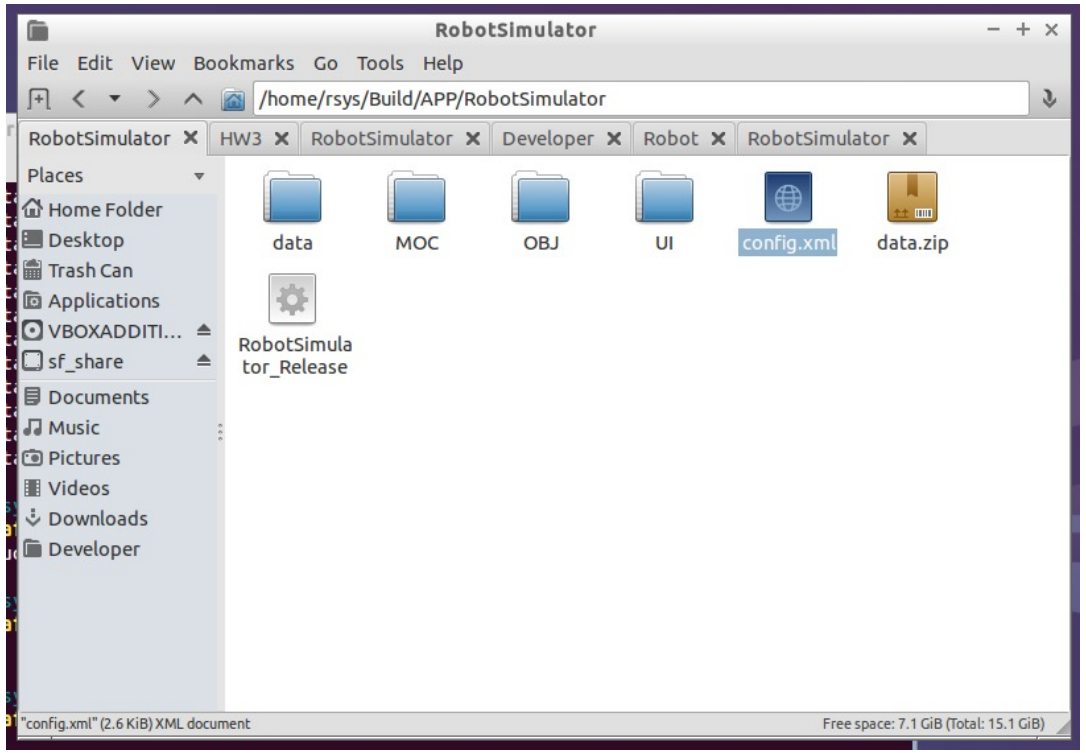


*确保顶层和底层在同一模式下进行编译；
活动工程总是在这里显示，可以以此来判断编译的是哪个工程

*Checkpoint 4

4. XML

#1:运行一次RobotSimulator　　执行Open-Close操作
#2:进入Build/APP/RobotSimulator查看XML文件
#3:数据包拷贝至XML所在的文件夹并解压



#4:修改XML内容:

①datapath

```
<Macro>
  <DataPath>data</DataPath>
</Macro>
```

②Filename（改为实际使用的数据文件名）

```
<colorFilename>
  <Default>20171104_142403_667_color</Default>
</colorFilename>
<depthFilename>
  <Default>20171104_142403_668_depth</Default>
<filename>
  <Default>20171104_142403_807.lms</Default>
</filename>
 <filename>
   <Default>20171104_142403_809.odom</Default>
 </filename>
 <distancePerPulse>
   <Default>1</Default>
 </distancePerPulse>
```

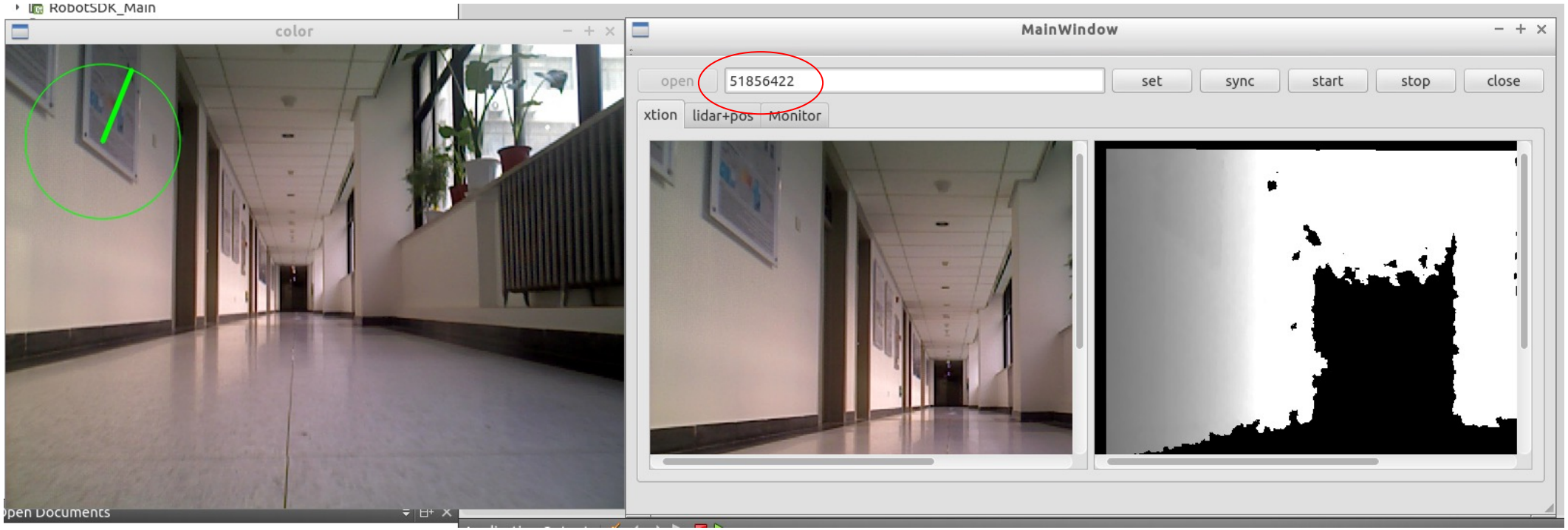
③DistancePerPulse（根据提供的文件计算）
④PixelSize（可选）

#5:保存退出

*Checkpoint 5

5.Run

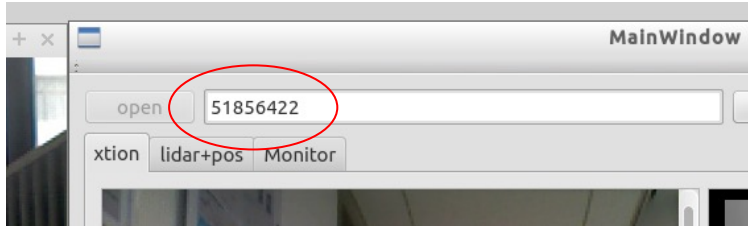
#1:重新运行RobotSimulator
按顺序执行Open-Set-Sync（需要等待）-Start



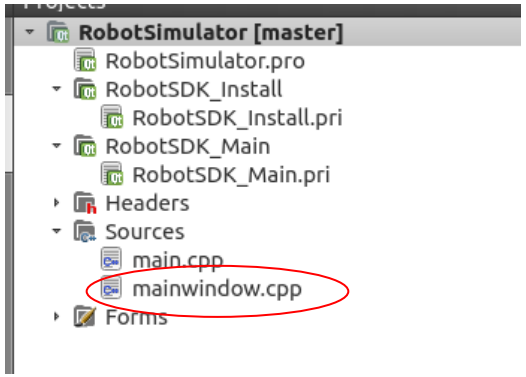
5.Run

#2.红圈代表初始时间戳，可以自行指定为数据有效部分的起始时间戳(数据文件:timestamp.log)

- 可以在界面中修改



- 或直接在RobotSimulator 中的 mainwindow.cpp中修改




```
ui->setupUi(this);
ui->starttime->setText("51856422");
```

#3.如果需要调整播放速度可在参考example_app中的mainwindow.cpp中修改RobotSimulator对应代码

```
//第六个speed是模拟速率
Simulator *EncoderSim = new Simulator("example1_module","Sensor_Encoder","EncoderSim","config.xml",QTime(),speed);//输入都有详细注释，使用IDE的话会自动提示
EncoderSim->setOutputNodesName(QStringList()<<"Deadreckoning");//定义好输出给哪个节点，QStringList里面存储的是后续节点的标示符
```

example_app代码

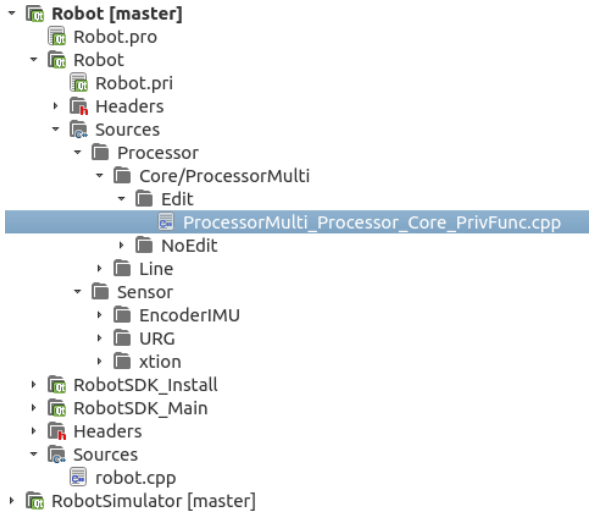
6.Details

#1:供修改的

- Robot/Sources/Processor/Core|ProcessorMulti/Edit/ProcessorMulti_Processor_Core_PrivFunc.cpp

#2:修改调试:

- 根据传感器的输入
  - 确定小车的行为
  - 根据行为控制小车
    - 计算Speed（或赋予一个定值）
    - 计算Steer（左-右+）（需要考虑事先计算的Steer与实际角度的比例）




```
//inputdata_0                                                   // EncoderIMU
//inputdata_1                                                   // URG
//inputdata_2                                                   // Xtion (RGB && depth)
//cv::imshow("color", inputdata_2.front()->cvColorImg);         // Show RGB image
//cv::imshow("depth", inputdata_2.front()->cvDepthImg);         // Show depth image

short steer = 100;              // [-400, 400]left  right
short speed = 100;              // [-180, 180]
```

传感器输入

如何确定传感器数据含义?

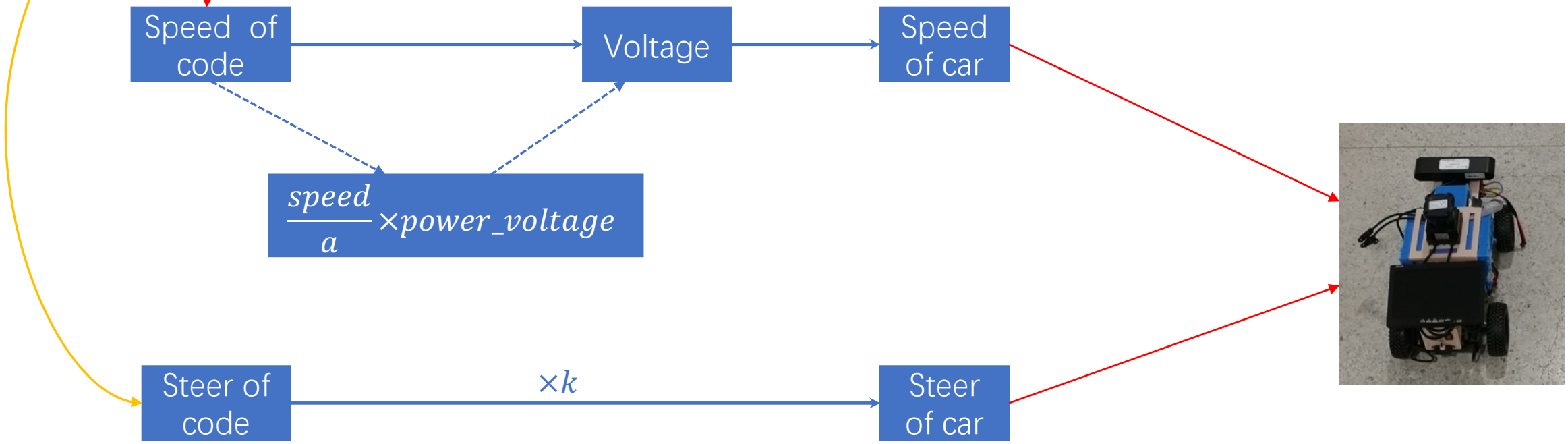
参考代码注释和头文件注释

6.Details

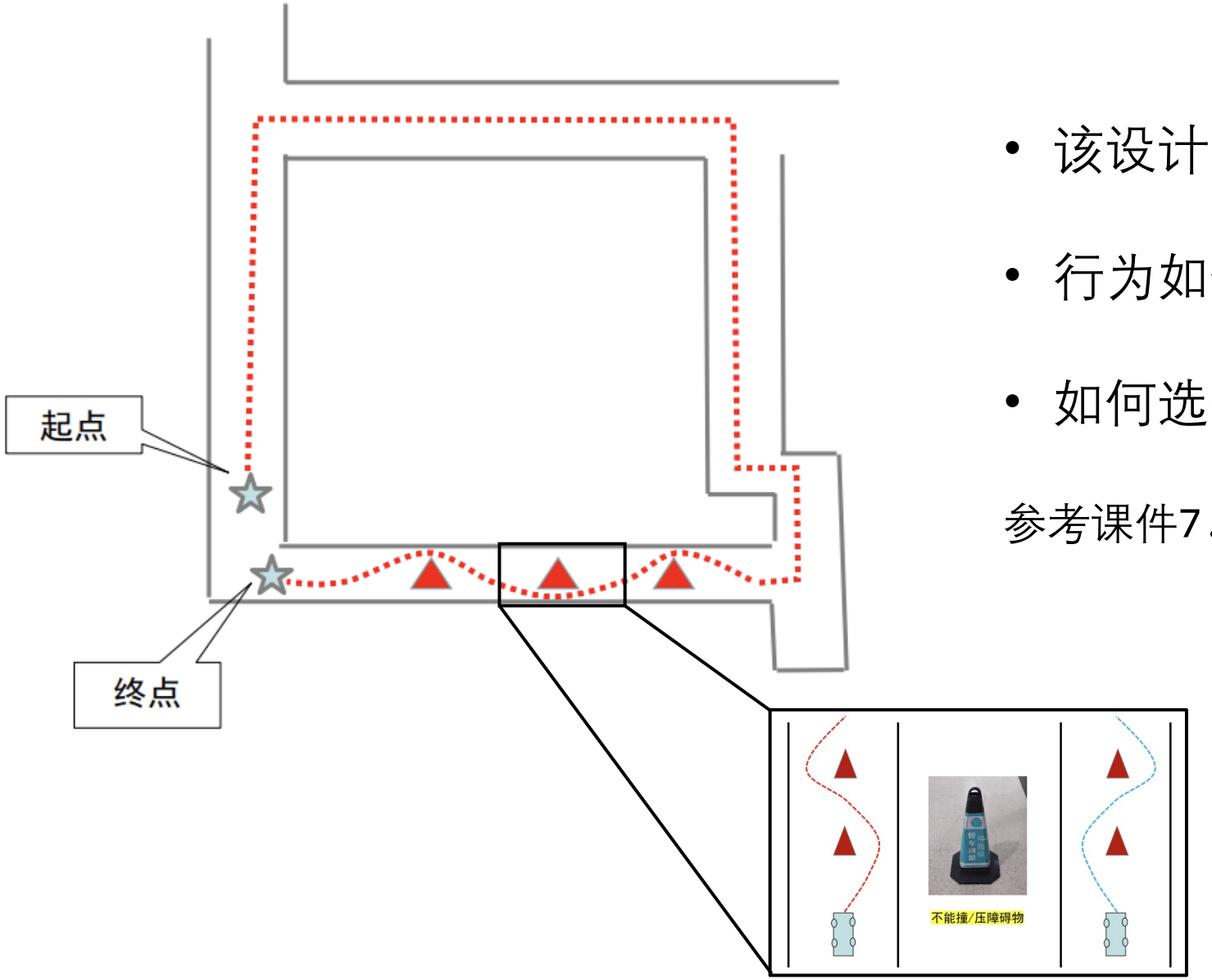
从代码到现实

```
//    vars->lastpulsenum = 0;

    short speed = 100;  //speed=[-180.+180]
    short steer = 0;    //steer=[-400,+400]
```



7.Behavior



- 该设计哪些行为?
- 行为如何触发?
- 如何选择当前的行为?

参考课件7.机器人控制架构

8.Control

- 确定行为后，需要根据行为控制小车。

  通过串口将数据发送到底层硬件，实现小车的控制

  封装形式：以两个short型变量实现
  speed：[-180,180],对应后退到前进
  steer：[-400，400]，对应舵机打到最左和舵机打到最右

  下发给底层（已包含在框架中）：

```
dataput[0] = 0xF8;
dataput[1] = 4;
*(short*)&dataput[2] = (short)steer;
*(short*)&dataput[4] = (short)speed;
dataput[6] = 0x8F;
```

- 如何确认speed 和 steer的数值？

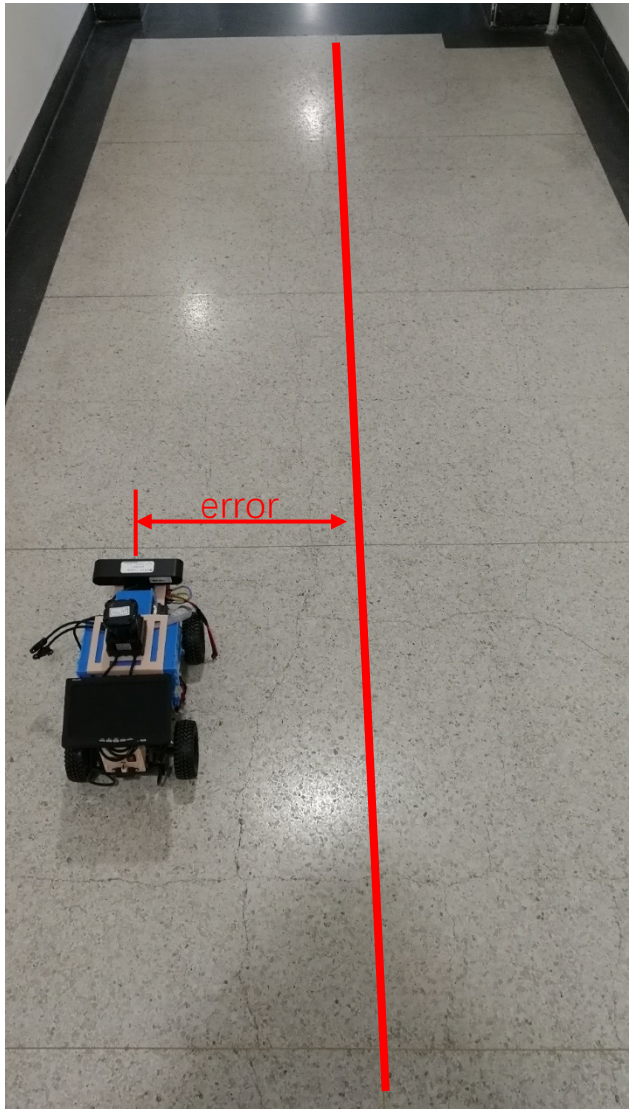
8.Control-PID

以steer 为例:

通过控制steer来让小车沿中线行驶

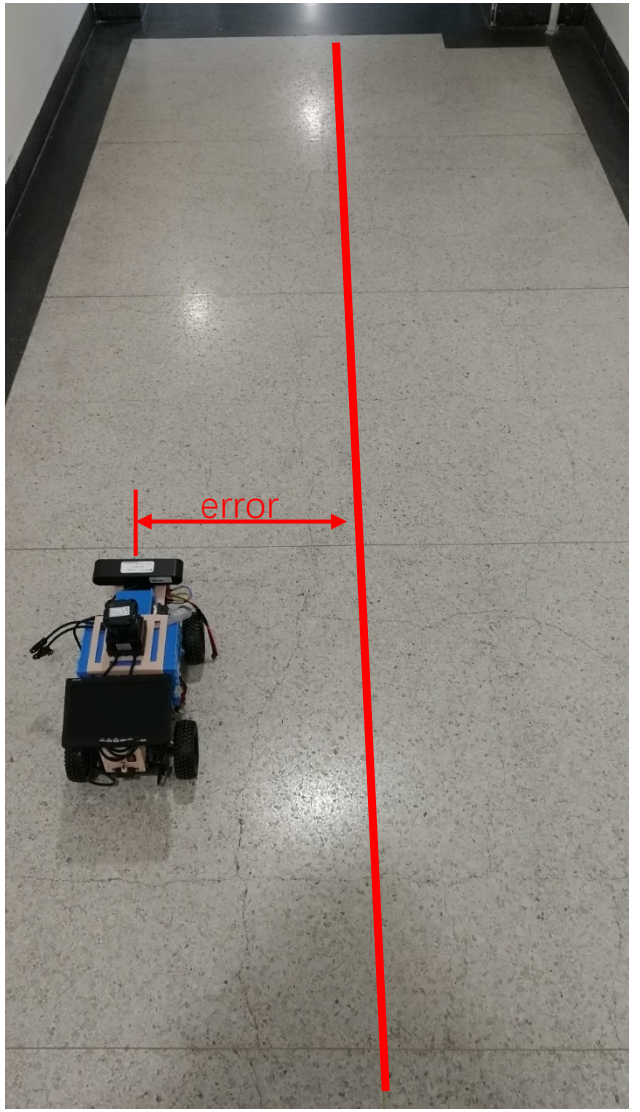
思路:

从error出发，得到我们需要的steer



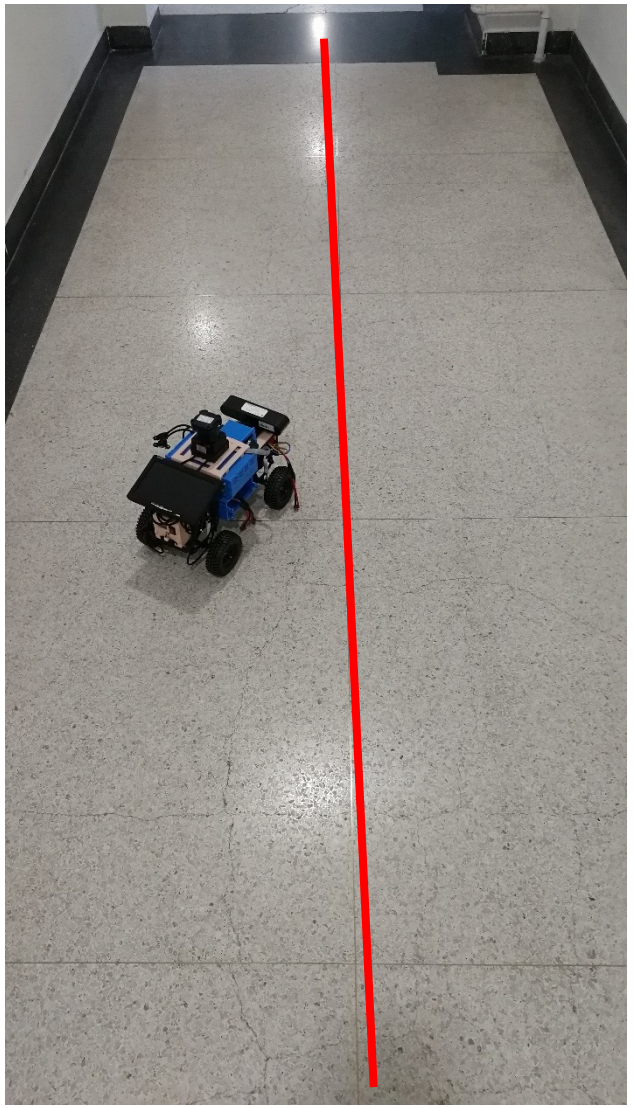
控制（P）

$$Steer_t = P \times error_t$$



控制（P）

$$Steer_t = P \times error_t$$

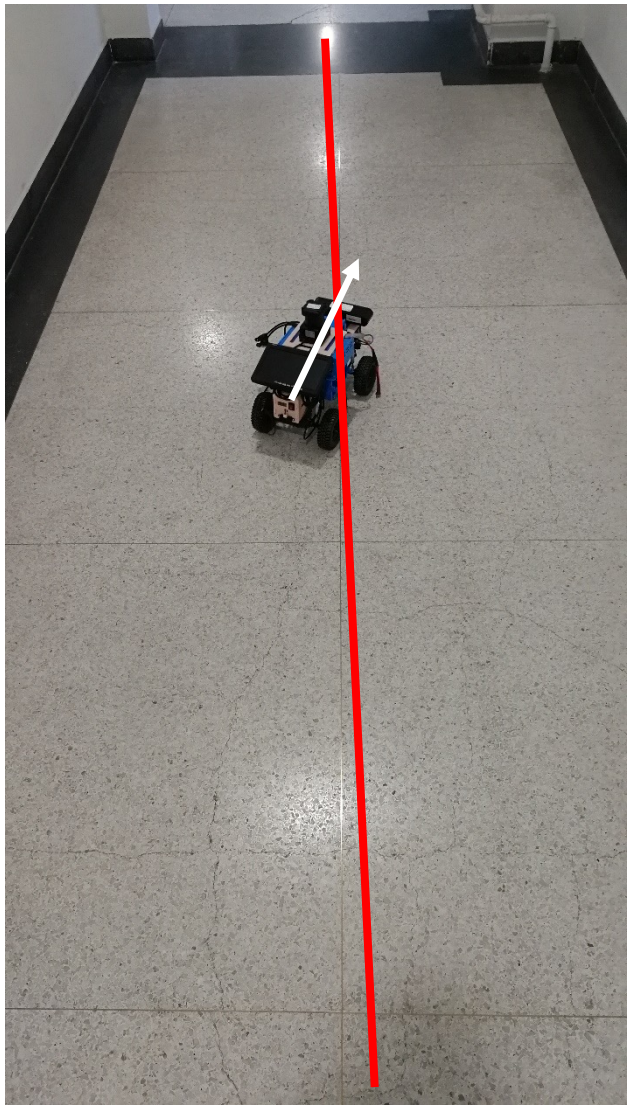


控制（P）

$$Steer_t = P \times error_t$$

$$error_t = 0$$

$$Steer_t = P \times error_t = 0$$



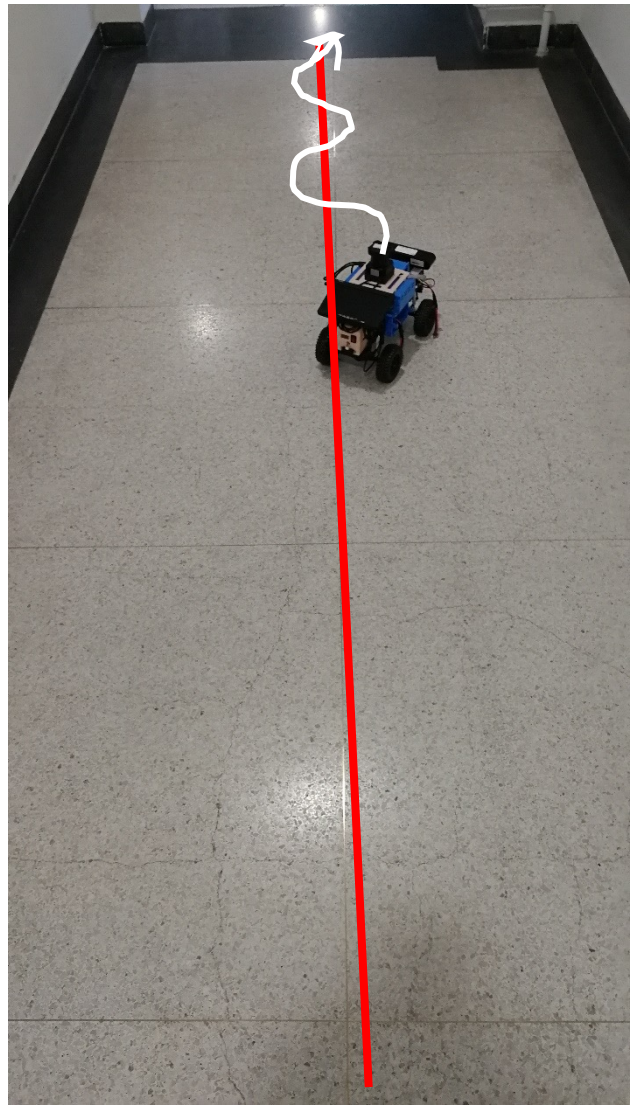
控制（P）

解决办法：让steer提前变小，别转得太猛

$$Steer_t = P \times error_t$$

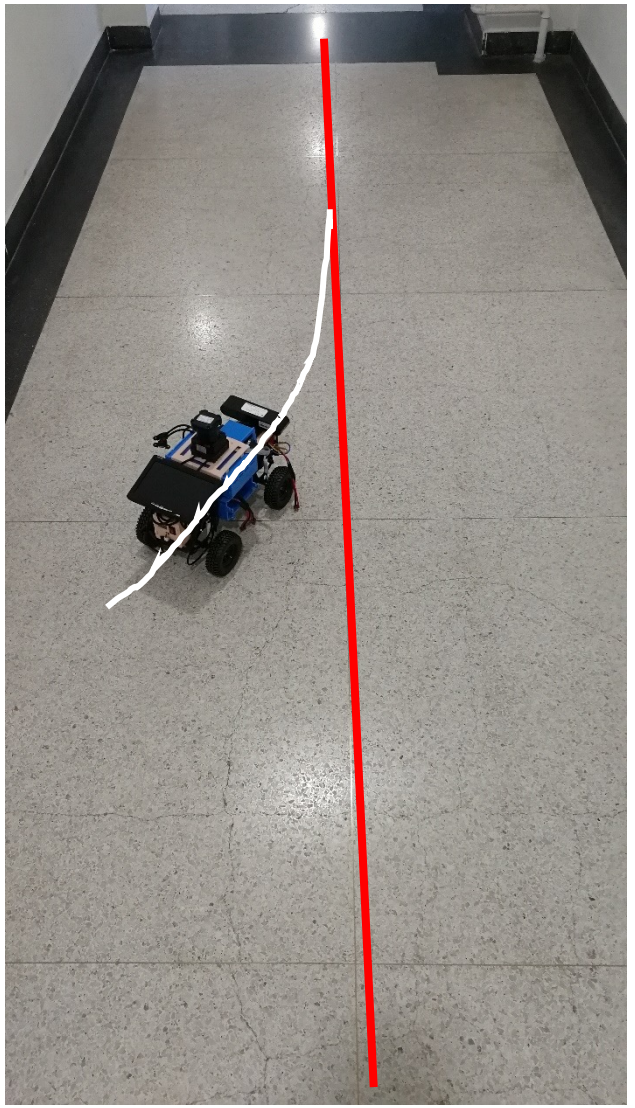
$$error_t = -k, (k > 0)$$

$$Steer_t = P \times (-k) = -kP$$



控制（P+D）

$$Steer_t = P \times error_t + D \times (error_t - error_{t-1})$$



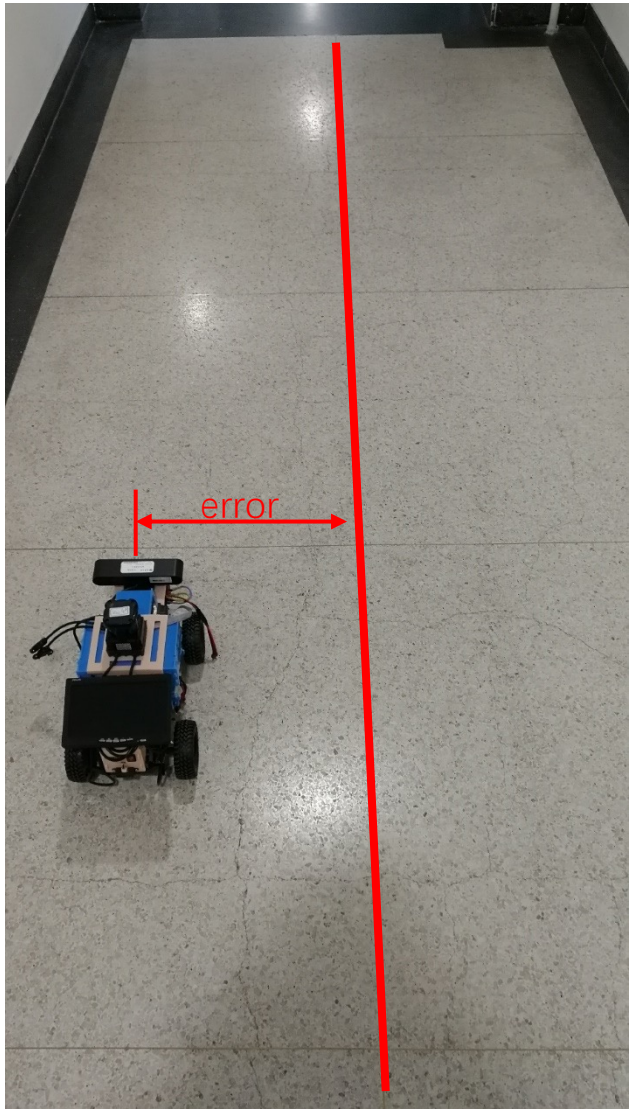
控制（P+D+I）

假如error很小，使舵机输出量不足以克服地面摩擦力

$$Int_t = Int_{t-1} + I \times error_t$$

$$Steer_t = P \times error_{\ t} + D \times (error_{\ t} - error_{\ t-1}) + Int_t$$

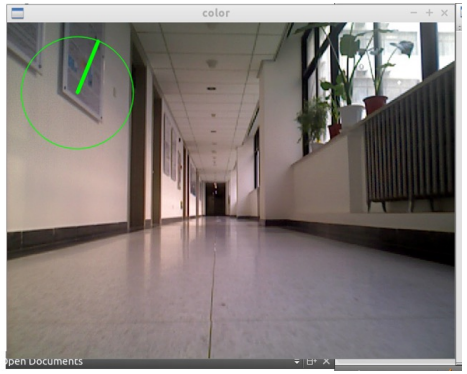
- P：快速到达控制目标
- I：消除累积误差
- D：减轻系统抖动
- 一般来讲，先调P，等车辆方向振荡不是太大时，稍微给点D消除抖动。
- 注意：P、D、I的正负和error的定义有关



尽量使用RobotSimulator完成代码的调试：

- 在本地写好算法，利用RobotSimulator完成算法逻辑是否正确：
  - 小车行为：
    - 输出当前行为，结合可视化中小车位置和信息输入，判断代码是否正确
  - 小车控制：
    - 可视化或输出Steer或Speed，验证代码逻辑是否正确

steer



- 注意：每次对底层程序进行修改都需要重新编译

9.Task

#0:完成标定计算任务，从而能使用本地模拟器进行模拟

#1:按照Behavior-based基于行为的机器人架构进行系统设计

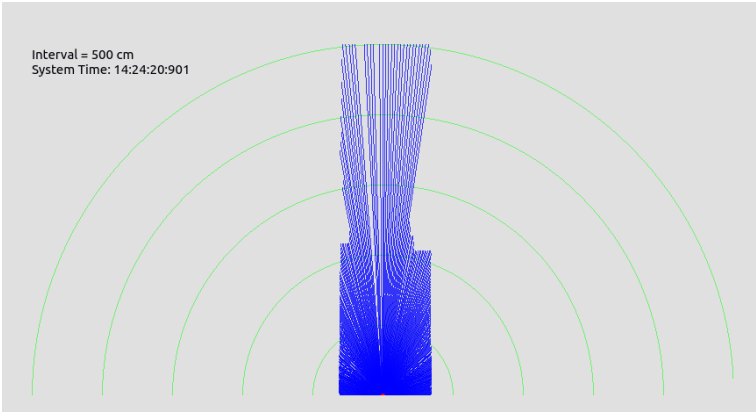
任务要点：设计合理的行为集、触发机制、行为选择方案

设计报告提交截止时间：11月12日(周日)

#2:绕理科二号楼安全行驶一圈

任务要点：寻路、障碍物检测与绕障控制

比赛时间：11月28日(周二)



9.Task

11月28日比赛前，每组有对于两个任务的实车测试

在群内选择对应助教的可选时间

实车测试注意事项：

- 每组两次，一次一个小时，严格控制时长。
- 充分利用本地的模拟调试代码，避免现场大幅度修改
- 留好参数接口，提前想好实车测试流程，方便现场修改
- 爱护车辆，严格避免碰撞

Q&A

参考

增量式PID：PID输出的不是直接控制量，而是控制量的增量

以速度控制为例：

$$error_t = target\_speed - current\_speed$$

$$\Delta Speed_t = P \times (error_t - error_{t-1}) + I \times error_t + D \times (error_t - 2error_{t-1} + error_{t-2})$$

$$Speed_t = \Delta Speed_t + Speed_{t-1}$$

增量式PID和位置式PID的关系？优劣？