工程实践与科技创新 IV-E

Lecture 7: EKF SLAM 和 OpenCV 色彩识别

授课老师: 张晗、王景川 *2022 年春季*

1 EKF SLAM

SLAM 指同时定位与建图。在本课程中，所建的地图是由 “路标点” 构成的特征地图。我们期望使用 EKF 同时估计机器人的位姿和地图上路标点的位置。EKF SLAM 的整体流程与 EKF 定位基本相同，只是会有一些由状态向量扩维引起的变化。

具体地，对于由 n 个路标点表示的地图，其状态向量可以表示为:

$$\begin{aligned} X_t &= \Big[\underbrace{x_t, y_t, \theta_t}_{\text{robot's pose}}, \underbrace{m_{1,t}^x, m_{1,t}^y}_{\text{landmark 1}}, \ldots, \underbrace{m_{n,t}^x, m_{n,t}^y}_{\text{landmark n}} \Big]^T \\ &= [x_{r,t}, m_{1,t}, \ldots, m_{1,t}]^T. \end{aligned}$$

其状态空间表达式为:

$$X_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \\ m_{1,t+1} \\ \vdots \\ m_{n,t+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & I_2 & & \\ & & & & \ddots & \\ & & & & & I_2 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \theta_t \\ m_{1,t} \\ \vdots \\ m_{n,t} \end{bmatrix} + \Delta t \begin{bmatrix} \cos\theta_t & 0 \\ \sin\theta_t & 0 \\ 0 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} \nu_t \\ \omega_t \end{bmatrix}}_{u_t}}_{f(X_t, u_t)} + \underbrace{\begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}}_{G} \underbrace{\tilde{w}_t}_{w_t},$$

$$Y_{i,t} = \underbrace{\begin{bmatrix} \sqrt{(m_{k_i,t}^x - x_t)^2 + (m_{k_i,t}^y - y_t)^2} \\ \arctan 2(m_{k_i,t}^y - y_t, m_{k_i,t}^x - x_t) - \theta_t \end{bmatrix}}_{h_{i,t}(X_t)} + v_{i,t}, \quad i = 1, \cdots, n_t, \quad k_i \in \{1, \cdots, n\}$$

其中 $X_t \in \mathbb{R}^{2n+3}$，$\tilde{w}_t \in \mathbb{R}^3$，$\mathrm{cov}(v_{i,t}, v_{i,t}) = R$，$\mathrm{cov}(\tilde{w}_t, \tilde{w}_t) = \tilde{Q}_t$，则

$$\mathrm{cov}(w_t, w_t) = G\tilde{Q}_t G^T = \begin{bmatrix} \tilde{Q}_t & 0 \\ 0 & 0 \end{bmatrix} := Q_t$$

这里需要注意的是，t **时刻的观测模型** $h_t(X_t)$ **的维数是时变的**，会随着当前 t 时刻观测到的路标点数目的变化而变化。假设第 t 时刻机器人观测到了 n_t 个路标点，则 $Y_t \in \mathbb{R}^{2n_t}$。除此之外，第 t 时刻观测到的第 i 个观测 $Y_{i,t}$ 所对应的路标点 $m_{k_i,t}$ 不一定是第 i 个路标点。在实现（尤其是计算 Jacobian 矩阵）的时候，需要尤其注意。

下面我们将对 EKF SLAM 的算法进行具体说明。

1.1 Predict Step

在 Predict Step, 我们的目标仍是基于系统的动态来预测状态向量和协方差。数学表达为:

$$\hat{X}_{t+1|t} = f(\hat{X}_{t|t}, u_t),$$
$$P_{t+1|t} = F_t P_{t|t} F_t^T + Q_t,$$

其中,

$$F_t = \frac{\partial f}{\partial X}|_{X=\hat{X}_{t|t}} = \begin{bmatrix} F_t^r & \mathbf{0}_{3\times 2n} \\ \mathbf{0}_{3\times 2n} & I_{2n\times 2n} \end{bmatrix}$$

其中，F_t^r 是机器人 unicycle 运动学模型的 Jacobi 矩阵，与上次课 EKF 定位中的推导的相同:

$$F_t^r = \begin{pmatrix} 1 & 0 & -v_t \sin\hat{\theta}_{t|t}\Delta t \\ 0 & 1 & v_t \cos\hat{\theta}_{t|t}\Delta t \\ 0 & 0 & w_t \Delta t \end{pmatrix}$$

最终，协方差的预测公式为:

$$\begin{aligned} P_{t+1|t} &= F_t P_{t|t} F_t^T + Q_t \\ &= \begin{bmatrix} F_t^r & \mathbf{0}_{3x2n} \\ \mathbf{0}_{3x2n} & I_{2nx2n} \end{bmatrix} \begin{bmatrix} \Sigma_{rr} & \Sigma_{rm} \\ \Sigma_{mr} & \Sigma_{mm} \end{bmatrix} \begin{bmatrix} F_t^{rT} & \mathbf{0}_{3\times 2n} \\ \mathbf{0}_{3\times 2n} & I_{2n\times 2n} \end{bmatrix} + \begin{bmatrix} \tilde{Q}_t & 0 \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} F_t^r \Sigma_{rr} F_t^{rT} + \tilde{Q}_t & F_t^r \Sigma_{rm} \\ (F_t^r \Sigma_{rm})^T & \Sigma_{mm} \end{bmatrix} \end{aligned}$$

从最终的协方差预测公式也可看出，与机器人位姿相关的部分 $\Sigma_{rr}, \Sigma_{rm}, \Sigma_{mr}$ 有所变化，而与位姿无关的部分 Σ_{mm} 保持不变。

1.2 Update Step

在 Update Step, 我们的目标仍是基于观测信息来更新状态向量和协方差。数学表达为:

$$\begin{aligned} \hat{Y}_{t+1|t} &= h_t(\hat{X}_{t+1|t}) \\ K_t &= P_{t+1|t} H_t^T (H_t P_{t+1|t} H_t^T + R_t)^{-1}, \\ \hat{X}_{t+1|t+1} &= \hat{X}_{t+1|t} + K_t(Y_{t+1} - \hat{Y}_{t+1|t}), \\ P_{t+1|t+1} &= P_{t+1|t} - K_t H_t P_{t+1|t}. \end{aligned}$$

这里的观测噪声的协方差矩阵 R_t 是时变的，会随着 t 时刻观测到的路标点的个数的变化而变化。

1.2.1 计算预期观测

预期观测是指把当前的最佳估计 $\hat{X}_{t+1|t}$ 代入观测方程所得到的在当前最优估计下期望的观测量。这里的最佳估计，不仅包括机器人位姿的最佳估计，也包括地图路标点位置的最佳估计 (注意与 EKF 定位中地图路标点位置已知的情况的区别)，即:

$$\hat{X}_{t+1|t} = \left[\hat{x}_{t+1|t}, \hat{y}_{t+1|t}, \hat{\theta}_{t+1|t}, \hat{m}_{1,t+1}^T, \ldots, \hat{m}_{n,t+1|t}^T\right]^T$$

对于观测到的第 $i(i=1,\cdots,n_t)$ 个路标点，其预期观测为：

$$\begin{aligned}\hat{Y}_{i,t+1} &= h_{i,t}(\hat{X}_{t+1|t}) \\ &= \begin{bmatrix} \sqrt{(\hat{m}^x_{k_j,t+1|t}-\hat{x}_{t+1|t})^2+(\hat{m}^y_{k_j,t+1|t}-\hat{y}_{t+1|t})^2} \\ \arctan 2(\hat{m}^y_{k_j,t+1|t}-\hat{y}_{t+1|t},\hat{m}^x_{k_j,t+1|t}-\hat{x}_{t+1|t})-\hat{\theta}_{t+1|t} \end{bmatrix}\end{aligned}$$

我们还需要计算 Jacobian 矩阵 H_t。如上文中提到，第 t 时刻观测到的第 i 个观测 $Y_{i,t}$ 所对应的路标点 $m_{k_i,t}$ 不一定是第 i 个路标点；所以在计算 Jacobian 矩阵的时候，需要尤其注意：对状态向量 X_t 的元素求导的时候，哪些元素是 0？哪些元素不是 0?

1.3 新的路标点

有趣的是，如果机器人在 $t+1$ 时刻，发现之前从未观测过的路标点 $m_{n+1,t+1}$，会造成整个离散时间隐 Markov 链的状态维数发生变化：新的状态 X'_{t+1} 变为

$$X'_{t+1}=\begin{bmatrix} X_{t+1} \\ m_{n+1,t+1}\end{bmatrix}.$$

此时，从数学上严格地来说，之前设定的 Kalman 滤波器模型已经不再适用，因为根本的隐 Markov 链模型的维数已经发生了改变。但幸运的是，我们可以在 $t+1$ 时刻重新开始一个 Kalman 滤波器，模型与之前类似，但状态的维数增加了一维。其状态空间表达式为：

$$\begin{aligned}X'_{t+2} =& \underbrace{\begin{bmatrix} f(X'_{t+1},u_{t+1}) \\ m_{n+1,t+1}\end{bmatrix}}_{f'(X_{t+1},u_{t+1})} + \underbrace{\underbrace{\begin{bmatrix} G \\ 0\quad 0\quad 0\end{bmatrix}}_{G'}\tilde{w}_t}_{w'_t}, \\ Y_{t+1} =& h_t(X'_{t+1},u_{t+1})+v_t.\end{aligned}$$

其中观测模型 $h_t(X'_{t+1},u_{t+1})$ 的形式与之前保持不变；新的过程噪声的协方差相比于先前的 Q_t，变为：

$$\mathrm{cov}(w'_t,w'_t)=\begin{bmatrix} Q_t & 0 \\ 0 & 0\end{bmatrix}:=Q'_t$$

不失一般性，我们假设 $t+1$ 时刻观测到的新的路标点的序号为 n_{t+1}，即：

$$\begin{aligned}Y_{n_{t+1},1,t+1} &= \sqrt{(m^x_{n+1,t}-x_t)^2+(m^y_{n+1,t}-y_t)^2} \\ Y_{n_{t+1},2,t+1} &= \arctan 2(m^y_{n+1,t}-y_t,m^x_{n+1,t}-x_t)-\theta_t\end{aligned}$$

我们利用之前 t 时刻对 $t+1$ 时刻的估计，结合传感器观测，对这个新的 Kalman 滤波器初始化：

$$\hat{X}'_{t+1|t}=\begin{bmatrix} \hat{X}_{t+1|t} \\ \hat{m}^x_{n+1,t+1|t}:=Y_{n_{t+1},1,t+1}\cos(Y_{n_{t+1},2,t+1}+\hat{\theta}_{t+1|t}) \\ \hat{m}^y_{n+1,t+1|t}:=Y_{n_{t+1},1,t+1}\sin(Y_{n_{t+1},2,t+1}+\hat{\theta}_{t+1|t})\end{bmatrix}, P'_{t+1|t}=\begin{bmatrix} P_{t+1|t} & 0 \\ 0 & \tilde{P}\end{bmatrix}$$

其中 $\tilde{P}$ 可以取一个不太离谱的值。这里对 $\hat{X}'_{t+1|t}$ 和 $P'_{t+1|t}$ 的选择方式不完全严谨，但可以满足工程上的需求。

2 OpenCV 色彩识别介绍与应用

2.1 RGB 模型与 HSV 模型

数字图像处理中常用的采用模型是 RGB（红，绿，蓝）模型和 HSV（色调，饱和度，亮度）模型，RGB 广泛应用于彩色监视器和彩色视频摄像机，我们平时的图片一般都是 RGB 模型。而 HSV 模型更符合人描述和解释颜色的方式，HSV 的彩色描述对人来说是自然且非常直观的。

HSV 模型中颜色的参数分别是：色调（H：hue），饱和度（S：saturation），亮度（V：value）。由 A.R.Smith 在 1978 年创建的一种颜色空间, 也称六角锥体模型 (Hexcone Model)。

1. 色调（H：hue）：用角度度量，取值范围为 0° - 360°，从红色开始按逆时针方向计算，红色为 0°，绿色为 120°，蓝色为 240°。它们的补色是：黄色为 60°，青色为 180°，品红为 300°；
2. 饱和度（S：saturation）：取值范围为 0.0 ~ 1.0，值越大，颜色越饱和。
3. 亮度（V：value）：取值范围为 0(黑色) ~ 255(白色)

2.2 RGB 与 HSV 相互转换公式

2.2.1 RGB 转 HSV

设 (r, g, b) 分别是一个颜色的红、绿和蓝坐标，它们的值是在 0 到 1 之间的实数，令 $\beta = \max\{r,g,b\}$,$\alpha = \min\{r,g,b\}$。那么，在 HSV 空间下，其色彩坐标为

$$h = \begin{cases} 0^\circ & \alpha = \beta \\ 60^\circ \times \frac{g-b}{\beta-\alpha} + 0^\circ & \beta = r \ and \ g \geq b \\ 60^\circ \times \frac{g-b}{\beta-\alpha} + 360^\circ & \beta = r \ and \ g < b \\ 60^\circ \times \frac{b-r}{\beta-\alpha} + 120^\circ & \beta = g \\ 60^\circ \times \frac{r-g}{\beta-\alpha} + 120^\circ & \beta = b \end{cases}$$

$$s = \begin{cases} 0 & \beta = 0 \\ \frac{\beta-\alpha}{\beta} = 1 - \frac{\alpha}{\beta} & otherwise \end{cases}$$

$$v = \beta$$

2.2.2 HSV 转 RGB

HSV 转为 RGB 的公式如下：

$$C = V \times S \tag{1}$$

$$X = C \times (1 - |(H/60) mod 2 - 1|) \tag{2}$$

$$m = V - C \tag{3}$$

$$(R', G', B') = \begin{cases} (C, X, 0) & 0^\circ \leq H < 60^\circ \\ (X, C, 0) & 60^\circ \leq H < 120^\circ \\ (0, C, X) & 12^\circ \leq H < 180^\circ \\ (0, X, C) & 180^\circ \leq H < 240^\circ \\ (X, 0, C) & 240^\circ \leq H < 300^\circ \\ (C, 0, X) & 300^\circ \leq H < 360^\circ \end{cases} \tag{4}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255) \tag{5}$$

2.3 OpenCV 中色彩识别函数

2.3.1 OpenCV 简介

OpenCV 是一个开源的发行的跨平台计算机视觉库，可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。它轻量级而且高效，由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

2.3.2 颜色空间转换函数 cvtColor

cvtColor 函数是 OpenCV 里用于图像颜色空间转换，可以实现 RGB 颜色、HSV 颜色、HSI 颜色、lab 颜色、YUV 颜色等转换，也可以彩色和灰度图互转。

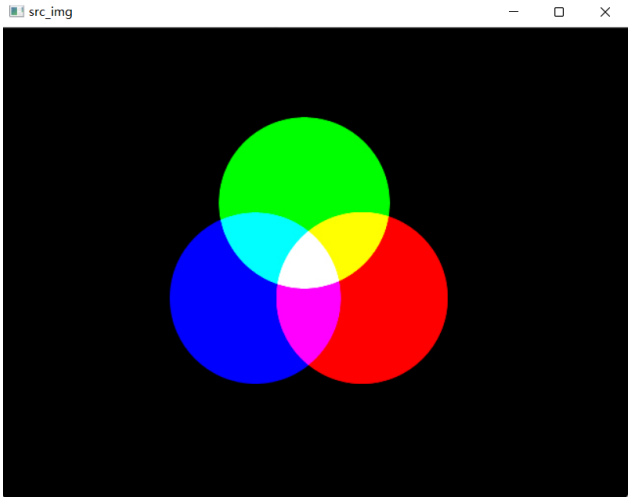
在识别不同颜色物体时，需要将图片转换到 HSV 色域中识别。OpenCV 中 cvtcolor 函数的定义如下：

```
dst = cv.cvtColor(src, code[, dst[, dstCn]])
# src:原图像  code: 颜色空间转换代码 dstCn: 目标图像通道数
```

如图所示，通过

```
hsv_image = cv2.cvtColor(src_image, cv2.COLOR_BGR2HSV)
```

可以将 RGB 空间中的图像转换到 HSV 中。



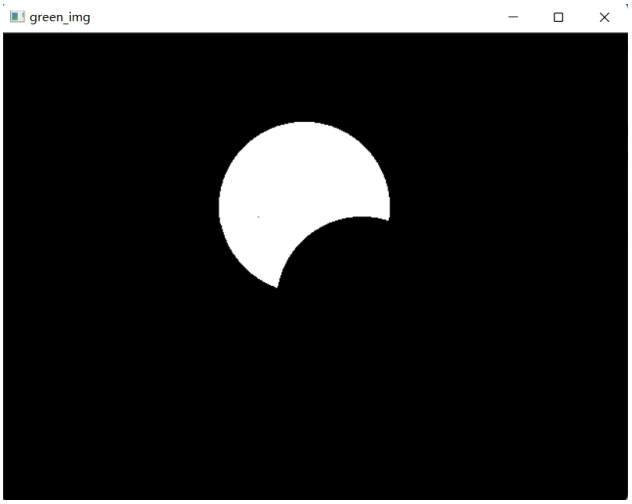
2.3.3 筛选目标颜色的函数 inRange

inRange 函数可以根据设定的阈值，去除阈值之外的背景部分，达到筛选某个颜色的效果。

```
dst = cv.inRange(src, lowerb, upperb[, dst])
#src: 原图    #lowerb: 下界    #upperb: 上界
```

该函数将图像内在阈值范围内的点置为 255，阈值外的点置为 0。
如图所示，通过

```
ball_color = 'green'
color_dist = {'red': {'Lower': np.array([0, 60, 60]),
                      'Upper': np.array([6, 255, 255])},
              'blue': {'Lower': np.array([100, 80, 46]),
              'Upper': np.array([124, 255, 255])},
              'green': {'Lower': np.array([35, 43, 35]),
              'Upper': np.array([90, 255, 255])},
            }
inRange_hsv = cv2.inRange(hsv_img, color_dist[ball_color]['Lower'],
color_dist[ball_color]['Upper'])
```

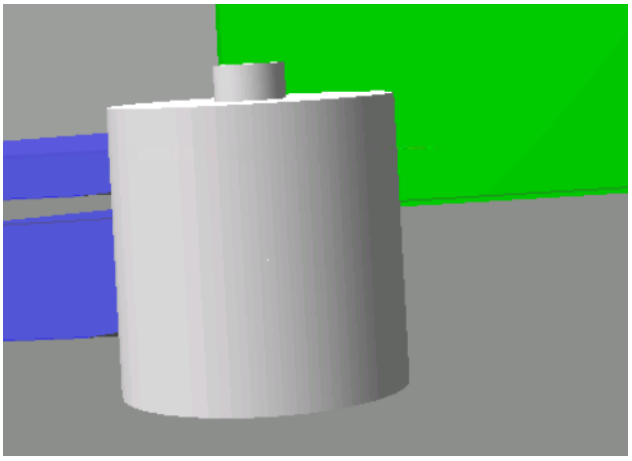


可以筛选出绿色的像素点。

3 在 Gazebo 上实现色彩辨识

3.1 为机器人搭建摄像头并发布图像消息

一、根据 Lecture 3 的附录文件，搭建具有摄像头的机器人模型如图所示：

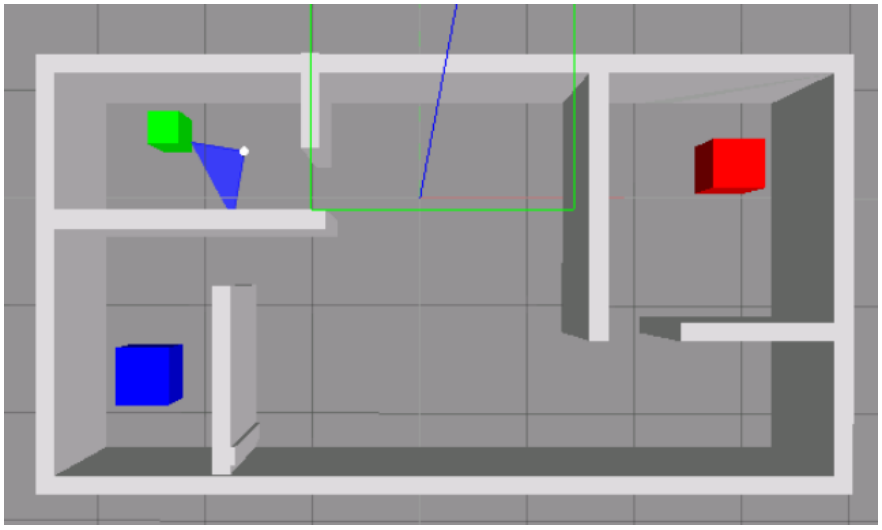


二、运行

```
$ roscore
$ rosrun gazebo_ros gazebo
```

进入 gazebo 界面

三、在环境中加入机器人模型并保存



四、编写 launch 文件如下：

```
<launch>
    <!-- 设置launch文件的参数 -->
    <arg name="world_name" value="$(find smartcar)/worlds/
```

```
    myhouse.world"/>
    <arg name="paused" default="false"/>
    <arg name="use_sim_time" default="true"/>
    <arg name="gui" default="true"/>
    <arg name="headless" default="false"/>
    <arg name="debug" default="false"/>
    <!-- 运行gazebo仿真环境 -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find smartcar)/worlds/
        myhouse.world" />
        <arg name="debug" value="$(arg debug)" />
        <arg name="gui" value="$(arg gui)" />
        <arg name="paused" value="$(arg paused)"/>
        <arg name="use_sim_time" value="$(arg use_sim_time)"/>
        <arg name="headless" value="$(arg headless)"/>
    </include>

    <!-- 运行joint_state_publisher节点，发布机器人的关节状态  -->
    <node name="joint_state_publisher" pkg="joint_state_publisher"
    type="joint_state_publisher" ></node>

    <!-- 运行robot_state_publisher节点，发布tf  -->
    <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="robot_state_publisher"  output="screen" >
        <param name="publish_frequency" type="double" value="50.0" />
    </node>

</launch>
```

五、关闭 gazebo 页面，运行

```
$ roslaunch smartcat camera.launch
```

打开环境

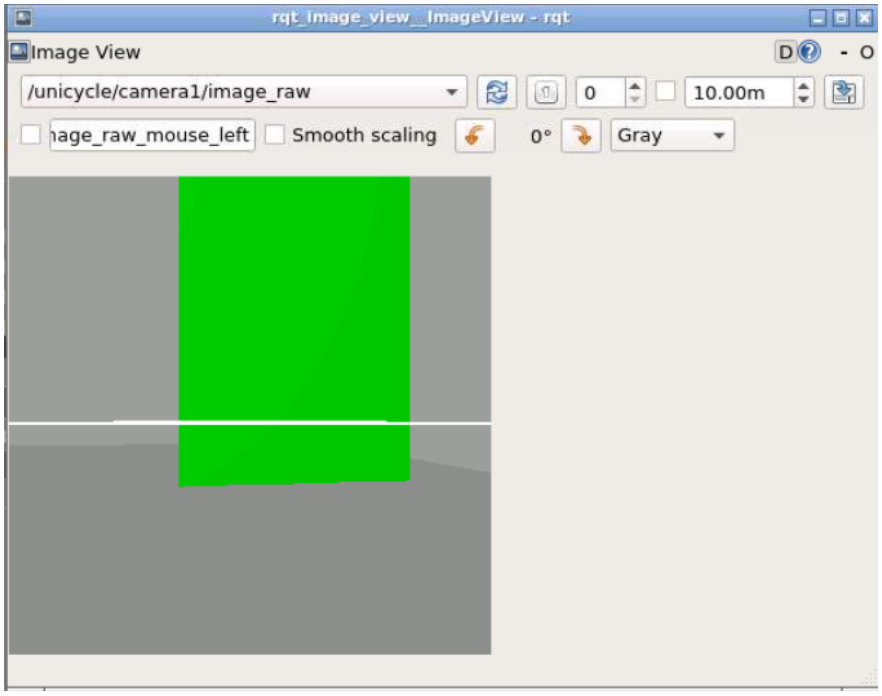
六、此时摄像头已经开始发布图像消息了，使用 rqt 工具查看当前机器人眼前的世界：（ROS 为我们提供的 Qt 工具箱里还包含一个图像显示的小工具，可以通过以下命令运行这个工具。）：在另一个终端内输入

```
$ rqt_image_view
```

查看摄像头拍摄的画面。也可以在 rviz 中添加 topic 查看。

七、编写 color_cube_detection.py 文件如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy
import cv2
from cv_bridge import CvBridge, CvBridgeError
```




```
from sensor_msgs.msg import Image
import numpy as np
from math import *
from geometry_msgs.msg import Pose

ball_color = 'green'

color_dist = {'red': {'Lower': np.array([0, 60, 60]),
'Upper': np.array([6, 255, 255])},
              'blue': {'Lower': np.array([100, 80, 46]),
              'Upper': np.array([124, 255, 255])},
              'green': {'Lower': np.array([35, 43, 35]),
              'Upper': np.array([90, 255, 255])},
            }

class Image_converter:
    def __init__(self):
        self.bridge = CvBridge()

        self.image_pub = rospy.Publisher('colorcube_detect',Image,
        queue_size = 10)

        self.image_sub = rospy.Subscriber('/unicycle/camera1/image_raw',
        Image,self.callback)
```

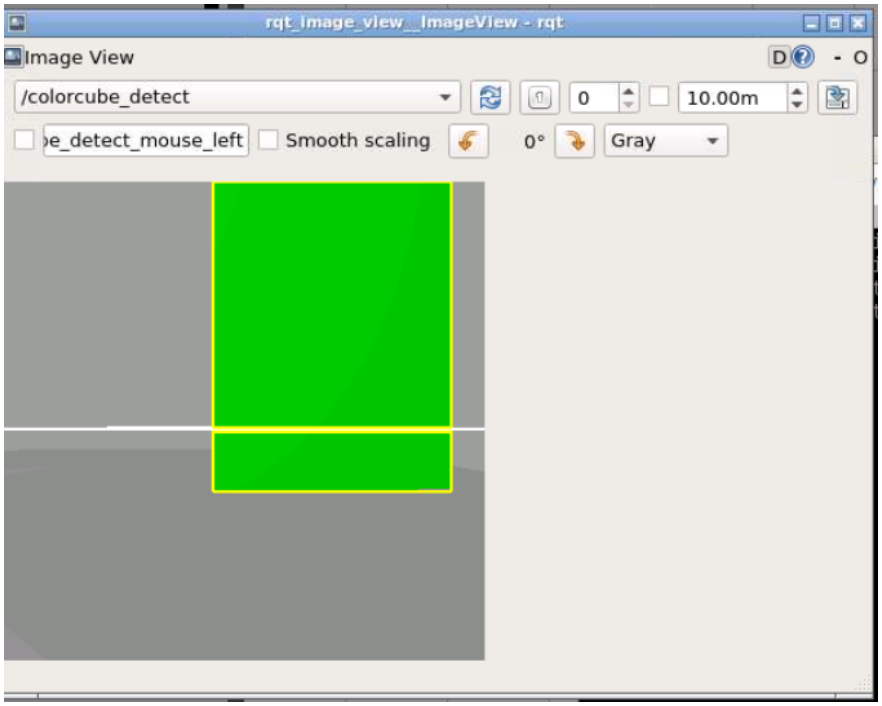
7: EKF SLAM 和 OpenCV 色彩识别-9

```
    def callback(self, data):
        try:
            #将相机数据转化为cv可以处理的np格式
            cv_image = self.bridge.imgmsg_to_cv2(data,"bgr8")

        except CvBridgeError as e:
                print e
        detect_image = self.detect_table(cv_image)
        try:
            #发布处理后的图片
            self.image_pub.publish(self.bridge.cv2_to_imgmsg
            (detect_image, "bgr8"))
        except CvBridgeError as e:
                print e

    def detect_table(self,image):
        #高斯滤波降噪
        g_image = cv2.GaussianBlur(image, (5, 5), 0)
        #RGB空间转为HSV空间
        hsv = cv2.cvtColor(g_image, cv2.COLOR_BGR2HSV)
        #腐蚀消除噪点
        erode_hsv = cv2.erode(hsv, None, iterations=2)
        #筛选指定颜色的区域
        inRange_hsv = cv2.inRange(erode_hsv, color_dist[ball_color]
        ['Lower'], color_dist[ball_color]['Upper'])
        #提取该区域的轮廓
        contours = cv2.findContours(inRange_hsv.copy(),
        cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
        for i in contours:
            x, y, w, h = cv2.boundingRect(i)
            #画图框出该区域
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255,255),3)
        return image

if __name__ == "__main__":
    rospy.init_node("vision_manager")
    rospy.loginfo("start")
    Image_converter()
    rospy.spin()
```

在终端内输入

```
python color_cube_detection.py
```

在 Image View 界面左上角的下拉菜单，可以看到当前系统中所有可显示的图像话题列表，选择/table_detect_test 话题。可以看到程序已经将绿色色块框出。



7: EKF SLAM 和 OpenCV 色彩识别-11