基于深度学习 YOLO 的目标识别

1 环境配置

1.1 ubuntu + ROS

1.2 libfreenect2 驱动安装

https://github.com/OpenKinect/libfreenect2/blob/master/README.md#installation

Note: Ubuntu 12.04 is too old to support. Debian jessie may also be too old, and Debian stretch is implied in the following.

- Download libfreenect2 source
- `git clone https://github.com/OpenKinect/libfreenect2.git`
- `cd libfreenect2`
- (Ubuntu 14.04 only) Download upgrade deb files
- `cd depends; ./download_debs_trusty.sh`
- Install build tools
- `sudo apt-get install build-essential cmake pkg-config`
- Install libusb. The version must be >= 1.0.20.
  i. (Ubuntu 14.04 only) `sudo dpkg -i debs/libusb*deb`
  ii. (Other) `sudo apt-get install libusb-1.0-0-dev`
- Install TurboJPEG
  i. (Ubuntu 14.04 and newer) `sudo apt-get install libturbojpeg libjpeg-turbo8-dev`
  ii. (Debian) `sudo apt-get install libturbojpeg0-dev`
- Install OpenGL
  i. (Ubuntu 14.04 only) `sudo dpkg -i debs/libglfw3*deb; sudo apt-get install -f; sudo apt-get install libgl1-mesa-dri-lts-vivid` (If the last command conflicts with other packages, don't do it.)
  ii. (Odroid XU4) OpenGL 3.1 is not supported on this platform. Use `cmake -DENABLE_OPENGL=OFF` later.
  iii. (Other) `sudo apt-get install libglfw3-dev`
- Install OpenCL (optional)

    - Intel GPU
        a. (Ubuntu 14.04 only) `sudo apt-add-repository ppa:floe/beignet; sudo apt-get update; sudo apt-get install beignet-dev; sudo dpkg -i debs/ocl-icd*deb`
        b. (Other) `sudo apt-get install beignet-dev`
        c. For older kernels, `# echo 0 >/sys/module/i915/parameters/enable_cmd_parser` is needed. See more known issues at https://www.freedesktop.org/wiki/Software/Beignet/.
    - AMD GPU: Install the latest version of the AMD Catalyst drivers from https://support.amd.com and `apt-get install opencl-headers`.
    - Mali GPU (e.g. Odroid XU4): (with root) `mkdir -p /etc/OpenCL/vendors; echo /usr/lib/arm-linux-gnueabihf/mali-egl/libmali.so >/etc/OpenCL/vendors/mali.icd; apt-get install opencl-headers`.
    - Verify: You can install `clinfo` to verify if you have correctly set up the OpenCL stack.
- Install CUDA (optional, Nvidia only):
    - (Ubuntu 14.04 only) Download `cuda-repo-ubuntu1404...*.deb` ("deb (network)") from Nvidia website, follow their installation instructions, including `apt-get install cuda` which installs Nvidia graphics driver.
    - (Jetson TK1) It is preloaded.
    - (Nvidia/Intel dual GPUs) After `apt-get install cuda`, use `sudo prime-select intel` to use Intel GPU for desktop.
    - (Other) Follow Nvidia website's instructions.
- Install VAAPI (optional, Intel only)
    . (Ubuntu 14.04 only) `sudo dpkg -i debs/{libva,i965}*deb; sudo apt-get install -f`
    i. (Other) `sudo apt-get install libva-dev libjpeg-dev`
    ii. Linux kernels 4.1 to 4.3 have performance regression. Use 4.0 and earlier or 4.4 and later (Though Ubuntu kernel 4.2.0-28.33~14.04.1 has backported the fix).
- Install OpenNI2 (optional)
    . (Ubuntu 14.04 only) `sudo apt-add-repository ppa:deb-rob/ros-trusty && sudo apt-get update` (You don't need this if you have ROS repos), then `sudo apt-get install libopenni2-dev`
    i. (Other) `sudo apt-get install libopenni2-dev`
- Build

```
cd ..
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
make
make install
```

You need to specify `cmake -Dfreenect2_DIR=$HOME/freenect2/lib/cmake/freenect2` for CMake based third-party application to find libfreenect2.

- Set up udev rules for device access: `sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/`, then replug the Kinect.
- Run the test program: `./bin/Protonect`
- Run OpenNI2 test (optional): `sudo apt-get install openni2-utils && sudo make install-openni2 && NiViewer2`. Environment variable `LIBFREENECT2_PIPELINE` can be set to `cl`, `cuda`, etc to specify the pipeline.

1.3 iai_kinect2 包安装

https://github.com/code-iai/iai_kinect2#install

依赖 Dependencies

- ROS Hydro/Indigo
- OpenCV (2.4.x, using the one from the official Ubuntu repositories is recommended)
- PCL (1.7.x, using the one from the official Ubuntu repositories is recommended)
- Eigen (optional, but recommended)
- OpenCL (optional, but recommended)
- libfreenect2 (>= v0.2.0, for stability checkout the latest stable release)

详细安装步骤

1. Install the ROS. Instructions for Ubuntu 14.04

   http://wiki.ros.org/indigo/Installation/Ubuntu

2. Setup your ROS environment

   http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

3. Install [libfreenect2]:

   Follow [the instructions] and enable C++11 by using `cmake ..` `-DENABLE_CXX11=ON` instead of `cmake ..`

   If something is not working, check out the latest stable release, for example `git checkout v0.2.0`.

4. Clone this repository into your catkin workspace, install the dependencies and build it:

cd ~/catkin_ws/src/ git clone https://github.com/code-iai/iai_kinect2.git cd iai_kinect2 rosdep install -r --from-paths . cd ~/catkin_ws catkin_make -DCMAKE_BUILD_TYPE="Release"

```
   *Note: `rosdep` will output errors on not being able to locate
`[kinect2_bridge]` and `[depth_registration]`.
   That is fine because they are all part of the iai_kinect2 package and `rosdep`
does not know these packages.*

   *Note: If you installed libfreenect2 somewhere else than in `$HOME/freenect2`
or a standard location like `/usr/local`
   you have to specify the path to it by adding
`-Dfreenect2_DIR=path_to_freenect2/lib/cmake/freenect2` to `catkin_make`.*
```

5. Connect your sensor and run `kinect2_bridge`:

```
```

roslaunch kinect2_bridge kinect2_bridge.launch

6. Calibrate your sensor using the `kinect2_calibration`. [Further details](kinect2_calibration#calibrating-the-kinect-one)

7. Add the calibration files to the `kinect2_bridge/data/<serialnumber>` folder. [Further details](kinect2_bridge#first-steps)

8. Restart `kinect2_bridge` and view the results using `rosrun kinect2_viewer kinect2_viewer kinect2 sd cloud`.

```
## GPU acceleration

### OpenCL with AMD

Install the latest version of the AMD Catalyst drivers from
https://support.amd.com and follow the instructions. Also install
`opencl-headers`.
```

sudo apt-get install opencl-headers

```
### OpenCL/CUDA with Nvidia

Go to
[developer.nvidia.com/cuda-downloads](https://developer.nvidia.com/cuda-dow
nloads) and select `linux`, `x86_64`, `Ubuntu`, `14.04`, `deb(network)`.
Download the file and follow the instructions. Also install `nvidia-modprobe`
and `opencl-headers`.
```

sudo apt-get install nvidia-modprobe opencl-headers

```
You also need to add CUDA paths to the system environment, add these lines to
you `~/.bashrc`:
```

export LD_LIBRARY_PATH="/usr/local/cuda/lib64:${LD_LIBRARY_PATH}" export PATH="/usr/local/cuda/bin:${PATH}"

```
A system-wide configuration of the libary path can be created with the following commands:
```

echo "/usr/local/cuda/lib64" | sudo tee /etc/ld.so.conf.d/cuda.conf sudo ldconfig

```
### OpenCL with Intel

You can either install a binary package from a PPA like [ppa:floe/beignet](https://launchpad.net/~floe/+archive/ubuntu/beignet), or build beignet yourself.
It's recommended to use the binary from the PPA.
```

sudo add-apt-repository ppa:floe/beignet && sudo apt-get update sudo apt-get install beignet beignet-dev opencl-headers

安装过程相关问题

1 If I have any question or someting is not working, what should I do first?

First you should look at this FAQ and the FAQ from libfreenect2. Secondly, look at issue page from libfreenect2 and the issue page of iai_kinect2 for similar issues and solutions.

2 Point clouds are not being published?

Point clouds are only published when the launch file is used. Make sure to start kinect2_bridge with `roslaunch kinect2_bridge kinect2_bridge.launch`.

3 Will it work with OpenCV 3.0

Short answer: No.

Long answer: Yes, it is possible to compile this package with OpenCV 3.0, but it will not work. This is because cv_bridge is used, which itself is compiled with OpenCV 2.4.x in ROS Indigo/Jade and linking against both OpenCV versions is not possible. Working support for OpenCV 3.0 might come with a future ROS release.

4 kinect2_bridge is not working / crashing, what is wrong?

There are many reasons why `kinect2_bridge` might not working. The first thing to find out whether the problem is related to `kinect2_bridge` or `libfreenect2`. A good tool for testing is `Protonect`, it is a binary located in `libfreenect2/build/bin/Protonect`. It uses libfreenect2 directly with a minimal dependency on other libraries, so it is a good tool for the first tests.
Execute:

- `./Protonect gl` to test OpenGL support.
- `./Protonect cl` to test OpenCL support.
- `./Protonect cpu` to test CPU support.

Before running `kinect2_bridge` please make sure `Protonect` is working and showing color, depth and ir images. If some of them are black, than there is a problem not related to `kinect2_bridge` and you should look at the issues from the libfreenect2 GitHub page for help.

If one of them works, try out the one that worked with `kinect2_bridge`: `rosrun kinect2_bridge kinect2_bridge _depth_method:=<opengl|opencl|cpu>`. You can also change the registration method with `_reg_method:=<cpu|opencl>`.

Protonect works fine, but kinect2_bridge is still not working / crashing.

If that is the case, you have to make sure that `Protonect` uses the same version of `libfreenect2` as `kinect2_bridge` does. To do so, run `make` and `sudo make install` in the build folder again. And try out `kinect2_bridge` again.

```
cd libfreenect2/build
make & sudo make install
```

Also make sure that you are not using OpenCV 3.0.

If it is still crashing, compile it in debug and run it with gdb:

```
cd <catkin_ws>
catkin_make -DCMAKE_BUILD_TYPE="Debug"
cd devel/lib/kinect2_bridge
gdb kinect2_bridge
// inside gdb: run until it crashes and do a backtrace
run
bt
quit
```

Open an issue and post the problem description and the output from the backtrace (`bt`).

kinect2_bridge hangs and prints "waiting for clients to connect"

2 文件结构

视觉部分主要涉及两个 ros pkg
darknet_ros – 完成视觉部分计算，得到目标所在空间三维坐标

calib　　-- 完成机械臂坐标系与照相机坐标系的转换

darknet_ros 目录 src 下主要文件说明

arm2cam.yml　-- 机械臂坐标系转换到照相机坐标系下的变换参数
cam2arm.yml　-- 照相机坐标系转换到机械臂坐标系下的变换参数

darknet_src/　-- darknet原生代码目录src
data/　　-- darknet原生数据目录data，
　　single.names存放目标标签
cfg/　　-- darknet原生配置目录cfg，
　　single.data 存放配置信息，如路径信息
　　tiny-yolo-single.cfg 存放模型网络配置信息

detector_ros.c
darknet_ros.cpp　-- 对darknet原生代码入口进行替换

calib目录src下主要文件说明
calib.cpp　　-- 坐标转换变换参数求取代码

3，开发说明

3.1 机械臂与照相机坐标转换

原理：calib 包接受来自机械臂末端在机械臂坐标系下的三维坐标， 同时记录机械臂末端在照相机图像坐标系下的像素坐标，采集多组样本，通过对应坐标点来拟合出坐标转换参数。

通过 ros 消息回调得到机械臂末端坐标

```
void pointCallback(const geometry_msgs::Point::ConstPtr &msg)
{
    pointLock.lock();
    armPoint.x = msg->x;
    armPoint.y = msg->y;
    armPoint.z = msg->z;
    pointLock.unlock();

}
```

通过在机械臂末端安装圆形黑点，来自动捕捉像素坐标

```
int image_process(cv::Mat& img, int *u, int *v)
{
    cv::Mat img_origin;
    img_origin = img;
```

```
cv::Mat edge;

int avg_u=0, avg_v=0, circle_count=0;

// Gaussian blur and extract edge
cv::cvtColor(img, img, CV_RGB2GRAY);
cv::GaussianBlur( img, img, cv::Size(9, 9), 2, 2 );
cv::Canny(img, edge, 10, 200, 3, false);


//Find circles and show centers
cv::vector<cv::Vec3f> circles0;
cv::HoughCircles(img, circles0, CV_HOUGH_GRADIENT, 1, 10000, 200, 20, 10, 20);

for( size_t i = 0; i < circles0.size(); i++ )
{
	cv::Point center0(cvRound(circles0[i][0]), cvRound(circles0[i][1]));
	int radius0 = cvRound(circles0[i][2]);
	// draw the circle center
	cv::circle( img_origin, center0, 3, cv::Scalar(0,255,0), -1, 8, 0 );//color -BGR
	// draw the circle outline
	cv::circle( img_origin, center0, radius0, cv::Scalar(0,0,255), 1.5, 8, 0 );

	avg_u += cvRound(circles0[0][0]);
	avg_v += cvRound(circles0[0][1]);
	circle_count++;
}
//printf("0000000000000\n");

cv::vector<cv::Vec3f> circles1;
cv::HoughCircles(img, circles1, CV_HOUGH_GRADIENT, 1, 10000,200, 20, 49, 52);
for( size_t i = 0; i < circles1.size(); i++ )
{
	cv::Point center1(cvRound(circles1[i][0]), cvRound(circles1[i][1]));
	int radius1 = cvRound(circles1[i][2]);
	// draw the circle center
	cv::circle( img_origin, center1, 3, cv::Scalar(0,255,0), -1, 8, 0 );
	// draw the circle outline
	cv::circle( img_origin, center1, radius1, cv::Scalar(255,0,0), 1.5, 8, 0 );
	avg_u += cvRound(circles1[0][0]);
	avg_v += cvRound(circles1[0][1]);
	circle_count++;
}
//printf("111111111111111\n");
```

```
cv::vector<cv::Vec3f> circles2;
cv::HoughCircles(img, circles2, CV_HOUGH_GRADIENT, 1, 10000, 200, 20, 30, 35);
for( size_t i = 0; i < circles2.size(); i++ )
{
	cv::Point center2(cvRound(circles2[i][0]), cvRound(circles2[i][1]));
	int radius2 = cvRound(circles2[i][2]);
	// draw the circle center
	cv::circle( img_origin, center2, 3, cv::Scalar(0,255,0), -1, 8, 0 );//color -BGR
	// draw the circle outline
	cv::circle( img_origin, center2, radius2, cv::Scalar(0,255,255), 1.5, 8, 0 );
	avg_u += cvRound(circles2[0][0]);
	avg_v += cvRound(circles2[0][1]);
	circle_count++;
}

cv::vector<cv::Vec3f> circles3;
cv::HoughCircles(img, circles3, CV_HOUGH_GRADIENT, 1, 10000, 200, 20, 40, 45);
for( size_t i = 0; i < circles3.size(); i++ )
{
	cv::Point center3(cvRound(circles3[i][0]), cvRound(circles3[i][1]));
	int radius3 = cvRound(circles3[i][2]);
	// draw the circle center
	cv::circle( img_origin, center3, 3, cv::Scalar(0,255,0), -1, 8, 0 );//color -BGR
	// draw the circle outline
	cv::circle( img_origin, center3, radius3, cv::Scalar(50,155,255), 1.5, 8, 0 );
	avg_u += cvRound(circles3[0][0]);
	avg_v += cvRound(circles3[0][1]);
	circle_count++;
}


//printf("333333333333333333, circlecount=%d\n", circle_count);

cv::namedWindow( "circles", 1 );
cv::imshow( "circles", img_origin);
cv::waitKey(5);

if(circle_count > 0) {
	*u = cvRound(avg_u/circle_count);
	*v = cvRound(avg_v/circle_count);
	return 0;
}
//printf("4444444444444444444\n");
```

```
        return -1;
    }
```

定义了几个按键，来执行交互：
s 键，程序记录三维坐标和图像二维坐标， 至少需要 4 组对应坐标
o 键，程序开始求解坐标变换参数
v 键，程序输出转换结果

```
    switch(key & 0xFF)
    {
    case 27:
    case 'q':
        running = false;
        break;
    case ' ':
    case 's':
            Points2D.push_back(cv::Point2f(u, v));
            pointLock.lock();
            Points3D.push_back(armPoint);
            pointLock.unlock();
            count++;

            std::cout<<"save point count: "<<count<<std::endl;
            std::cout<<"point2d: "<<cv::Point2f(u, v)<<std::endl;
            std::cout<<"point3d: "<<armPoint<<std::endl;
            break;
        case 'o':
            solveCameraPose();
            break;
        case 'v':
            kinect_point.at<double>(0,0) = X;
            kinect_point.at<double>(1,0) = Y;
            kinect_point.at<double>(2,0) = Z;
            kinect_point.at<double>(3,0) = 1.0;

            arm_point = cam2arm * kinect_point;
            std::cout<<"cam point: "<<kinect_point<<std::endl;
            std::cout<<"arm point: "<<arm_point<<std::endl;
            std::cout<<"Real arm point: "<<armPoint<<std::endl;

            break;
    }
```

3.2 darknet_ros 包裹及相关代码

darknet_ros.cpp 中接受 kinect v2 驱动传来的图像数据，借鉴 iai_kinect2\kinect2_viewer\src\view.cpp 源代码完成图像数据读取

路径配置

```
char datacfg[] = "/home/robot/catkin_ws/src/darknet_ros/src/cfg/single.data";
char cfgfile[] = "/home/robot/catkin_ws/src/darknet_ros/src/cfg/tiny-yolo-single.cfg";
char weightfile[] = "/home/robot/catkin_ws/src/darknet_ros/src/tiny-yolo-single.weights";
```

话题配置

```
const std::string CAMERA_COLOR_TOPIC = "/kinect2/qhd/image_color_rect";
const std::string CAMERA_DEPTH_TOPIC = "/kinect2/qhd/image_depth_rect";
```

Receiver 类构造函数中， 加载照相机坐标转换到机械臂坐标系下的变换参数

```
        cam2arm = cv::Mat::zeros(4, 4, CV_64FC1);
        cv::FileStorage fs("/home/robot/catkin_ws/src/darknet_ros/src/cam2arm.yml",
cv::FileStorage::READ);
        fs["R_t"] >> cam2arm;
        fs.release();
```

消息回调函数，接收所要识别目标的编号，通过 DETECT_FLAG 标志，来控制图像接收回调是否进行目标检测任务

```
    void ugvReceiveCallback(const id_data_msgs::ID_Data::ConstPtr &msg)
    {
        if(msg->id == 3 &&  msg->data[0]== 1) {
            ROS_INFO("MSG=======%d\n", msg->id);
            //int flag = msg->data[0];
            object_row = msg->data[1];
            object_col = msg->data[2];

            DETECT_FLAG = 1;
            car_adjust = CAR_ADJUST_NUM;

            id_data_msgs::ID_Data feedback;
            feedback.id = 3;
            feedback.data[0] = 14;
            result_pub.publish(feedback);
        }
    }
```

目标检测

void objectDetect(cv::Mat c_image, cv::Mat d_image)

接收 rgb 图像和深度图，完成目标的检测和定位

```
void objectDetect(cv::Mat c_image, cv::Mat d_image)
{
    image im = Mat_to_image(c_image);
    cvtColor(c_image, c_image, CV_RGB2BGR);

    cv::Mat color_image = c_image.clone();
    cv::Mat depth_image = d_image.clone();
    createCloud(depth_image, color_image,    cloud);
    //          pcl::visualization::PCLVisualizer::Ptr                  visualizer(new
pcl::visualization::PCLVisualizer("Cloud Viewer"));
    //          const std::string cloudName = "rendered";
    //          visualizer->addPointCloud(cloud, cloudName);
    //
    visualizer->setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_SIZ
E, 1, cloudName);
    //          visualizer->initCameraParameters();
    //          visualizer->setBackgroundColor(0, 0, 0);
    //          visualizer->setPosition(mode == BOTH ? color.cols : 0, 0);
    //          visualizer->setSize(color.cols, color.rows);
    //          visualizer->setShowFPS(true);
    //          visualizer->setCameraPosition(0, 0, 0, 0, -1, 0);
    //          visualizer->registerKeyboardCallback(&Receiver::keyboardEvent, *this);
    //          visualizer->updatePointCloud(cloud, cloudName);
    //          for(; running && ros::ok();)
    //          {
    //              visualizer->spinOnce(10);
    //          }
    //          visualizer->close();

    //float *results = GetBoxDistance(cloud, 0 + 0.05, 0, 0);



    printf("Need obj: row = %d, col = %d\n", object_row, object_col);
    printf("predict start\n");
    boxes = test_detector_ros(net, im, names, alphabet, thresh, hier_thresh);
    //printf("predict over\n");

    // get the number of bounding boxes found
```

```
        int num = boxes[0].num;
        printf("################ boxes num: %d######################\n", num);

        //print_boxes(boxes, num);

        int i;
        int u, v;
        float X, Y, Z;

        std::vector<Obj_loc> obj_loc;
        std::vector<Obj_loc> obj_row;

        float robot_height_groud = 0.34;   //unit: m
        float shelf_height_groud = 0.26;
        float shelf_grid_width=0.3467;
        float shelf_grid_height=0.21;
        float shelf_grid_gap = 0.04;

        float bound_left, bound_right, bound_low, bound_high;
        int obj_index=-1;

        //float *results = GetBoxDistance(cloud, obj_loc[obj_index].x, obj_loc[obj_index].y, obj_loc[obj_index].z);

        bound_left = shelf_grid_width/2;
        bound_right = -shelf_grid_width/2;
        bound_low = shelf_height_groud - robot_height_groud + (object_row - 1)*(shelf_grid_height + shelf_grid_gap);
        bound_high = bound_low + shelf_grid_height;

        int valid_count = 0;

        printf("bound: %f, %f, %f, %f\n", bound_right, bound_left, bound_low, bound_high);
        for(i=0; i<num; i++) {
            u = boxes[i].x;
            v = boxes[i].y;

            int x = boxes[i].x - boxes[i].w/2;
            int y = boxes[i].y - boxes[i].h/2;
            int w = boxes[i].w;
            int h = boxes[i].h;
            cv::Rect rect(x, y, w, h);

            int tu, tv;
```

```
            Z = get_depth_and_center(depth_image, rect, &tu, &tv);
            if(Z<=0) {printf("z <=0\n"); continue;}
            UVtoXY(&X, &Y, u, v, Z);

            //convert kinect point to arm point
            cv::Mat kinect_point = cv::Mat::zeros(4, 1, CV_64FC1);
            kinect_point.at<double>(0,0) = X;
            kinect_point.at<double>(1,0) = Y;
            kinect_point.at<double>(2,0) = Z;
            kinect_point.at<double>(3,0) = 1.0f;

            cv::Mat arm_point = cam2arm * kinect_point;

            Obj_loc o;
            o.x = arm_point.at<double>(0,0);
            o.y = arm_point.at<double>(1,0);
            o.z = arm_point.at<double>(2,0);
            obj_loc.push_back(o);


            //if(o.z >= bound_low && o.z<=bound_high && o.y>=bound_right &&
o.y<=bound_left) obj_index=i;
            if(o.z >= bound_low && o.z<=bound_high && o.y>=bound_right &&
o.y<=bound_left) obj_index=valid_count;


            //if(o.z >= bound_low && o.z<=bound_high) obj_row.push_back(o);

            printf("------ obj#%d ------\n", valid_count);
            //printf("class:%d, %s\n", boxes[i].Class, names[boxes[i].Class]);
            printf("center pixel: u=%d, v=%d\n", u, v);
            printf("cam coord: X=%.2fm, Y=%.2fm, Z=%.2fm\n", X, Y, Z);
            printf("arm coord: X=%.2fm, Y=%.2fm, Z=%.2fm\n", o.x, o.y, o.z);
            printf("\n");
            //              geometry_msgs::PointStamped target_point;
            //              target_point.header.frame_id = "targetPoint";
            //              target_point.header.stamp = ros::Time::now();
            //              target_point.point.x = arm_point.at<double>(0,0);
            //              target_point.point.y = arm_point.at<double>(1,0);
            //              target_point.point.z = arm_point.at<double>(2,0);
            //              point_pub.publish(target_point);
            valid_count++;
        }
```

```
        //cv::waitKey(0);

        if(obj_index>=0) {
            input_color_file = "color_" + std::to_string(object_row) + std::to_string(object_col)
+ ".png";
            input_depth_file = "depth_" + std::to_string(object_row) +
std::to_string(object_col) + ".png";

            //createCloud(color_image, depth_image, cloud);


            float *results = GetBoxDistance(cloud, obj_loc[obj_index].x, obj_loc[obj_index].y,
obj_loc[obj_index].z);


            printf("env info : L=%.2f, R=%.2f, T=%.2f, B=%.2f, X=%.2f\n", results[0], results[1],
results[2], results[3], results[4]);

            cv::imwrite("depth_image.png", depth_image);
            cv::imwrite(input_color_file.c_str(), color_image);
            cv::imwrite(input_depth_file.c_str(), depth_image);

            printf("========= Total %d obj find ==========\n", num);
            printf("Need obj: row = %d, col = %d\n", object_row, object_col);
            printf("Result:\n");
            printf("\tobj_index=%d\n", obj_index);
            printf("\tcoord: X=%.2fm, Y=%.2fm, Z=%.2fm\n", obj_loc[obj_index].x,
obj_loc[obj_index].y, obj_loc[obj_index].z);
            printf("\tfixed Y=%f\n", ((obj_loc[obj_index].y + results[0]) + (obj_loc[obj_index].y
- results[1]))/2.0);

            // tell arm where is object
            id_data_msgs::ID_Data arm_feedback;
            arm_feedback.id = 3;
            arm_feedback.data[0] = 27;
            arm_feedback.data[1] = int(100 * results[0]);
            arm_feedback.data[2] = int(100 * results[1]);
            arm_feedback.data[3] = int(100 * results[2]);
            arm_feedback.data[4] = int(100 * results[3]);
            arm_feedback.data[5] = int(100 * results[4]);
            result_pub.publish(arm_feedback);

            geometry_msgs::PointStamped target_point;
```

```
                target_point.header.frame_id = "targetPoint";
                target_point.header.stamp = ros::Time::now();

                target_point.point.x = obj_loc[obj_index].x;
                //target_point.point.y = obj_loc[obj_index].y;
                target_point.point.y = ((obj_loc[obj_index].y + results[0]) + (obj_loc[obj_index].y -
results[1]))/2.0 + 0.02;
                target_point.point.z = obj_loc[obj_index].z;
                point_pub.publish(target_point);

                // tell manager object detect successfully
                id_data_msgs::ID_Data feedback;
                feedback.id = 3;
                feedback.data[0] = 15;
                result_pub.publish(feedback);
                free(boxes);
                DETECT_FLAG = 0;
            }
            else {
                // tell manager object detect successfully
                id_data_msgs::ID_Data feedback;
                feedback.id = 3;
                feedback.data[0] = 13;
                result_pub.publish(feedback);
                free(boxes);
                DETECT_FLAG = 0;
            }

            printf("car_adjust=%d\n", car_adjust);
            cv::Mat ret = image_to_Mat(im);
            cv::imshow(DETECTION_WINDOW, ret);
            cv::waitKey(10);

            //free_image(im);
        }
```

4 操作说明

4.1 机械臂坐标转换

在机械臂末端安装黑色圆心，
1 启动 calib 包，
2 使用机械臂遥控器调整机械臂末端，
3 使用 s 键保存四组以上对应点，
4 使用 o 键求解得到变换参数文件
5 按 q 键退出
6 拷贝参数文件到 darknet_ros 包中

4.2 darknet_ros 目标检测模型训练

1 下载源代码

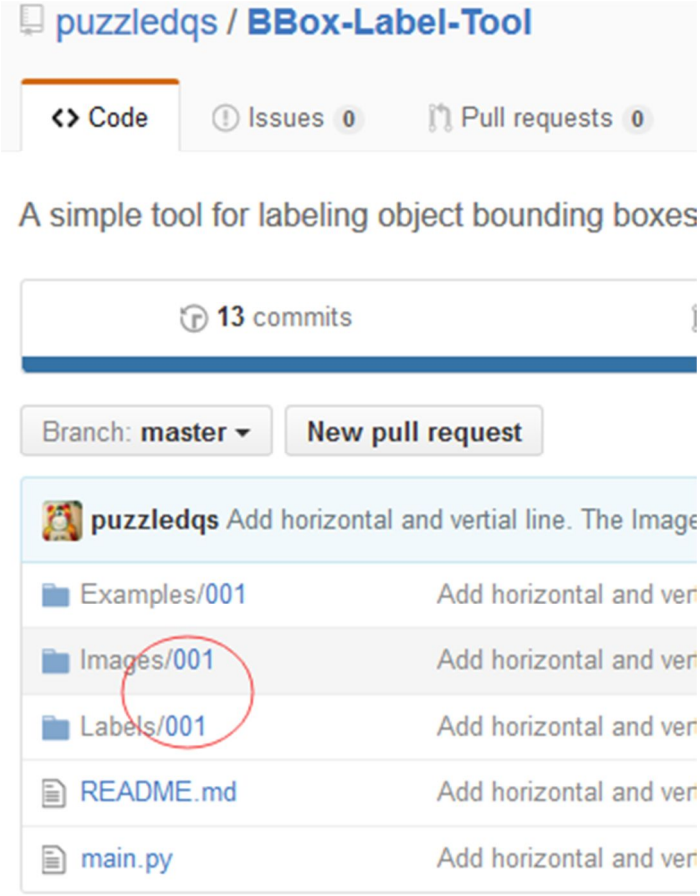
https://pjreddie.com/darknet/

https://github.com/pjreddie/darknet

2 数据集处理

（1） 获得数据集：我从 here 获取测数据集，实际上是二分类问题，检测的目标是 stopsign 和 yeildsign。
（2） 对数据及进行标注，用 BBox-Label-Tool 。然后修改代码。

（3） 改 BB 的代码：



Images 里面是要进行标注的图像，放在不同的文件夹里面，001，002，003 等等，每一个文件夹存放一类，如果你的类别是 cat 啊，bird 啊啥的，先把他改成 001，002 什么的， 然后标注完了再改回去。

具体标注方法参见
http://blog.csdn.net/qq_30401249/article/details/51504816

（4）生成坐标和类别的 txt 文件以及图片路径文件。
完成标注之后，在 label 文件夹下会显示标注后的与每张图片对应的.txt 文件，每个信息都有一下内容组成：

```
class_number
box1_x1 box1_y1 box1_width box1_height
box2_x1 box2_y1 box2_width box2_height
……
```

然后借助 darknet/scripts/convert.py
将其转化成程序中需要的格式：

```
class_number box1_x1_ratio box1_y1_ratio box1_width_ratio
box1_height_ratio
```

class_number box2_x1_ratio box2_y1_ratio box2_width_ratio box2_height_ratio
……
在 convert.py 的代码中，需要修改类别以适应不同类别的 label

```
""" Configure Paths"""

mypath = "labels/stopsign_original/" # 改

outpath = "labels/stopsign/" #改



cls = "stopsign" # 改

if cls not in classes:

    exit(0)

# 删除 cls_id = classes.index(cls)

# 改成如下

cls_id = 1 # 根据类别不同，改成不用的类标，与文件夹对应



wd = getcwd()

list_file = open('%s/%s_list.txt'%(wd, cls), 'w') # 存储图片绝对位置信息
```

例如：
转换前：
2
61 90 72 103
198 5 243 54

转换后：
0 0.123552123552 0.559278350515 0.0926640926641 0.10824742268
0 0.743243243243 0.585051546392 0.0579150579151 0.0773195876289

将生成的各种类别下的图片列表放到 training_list.txt 文件里
比如，有 stopsign_listing.txt, yeildsign_listing.txt 两类

```
cat stopsign_listing.txt* yeildsign_listing.txt > train.txt
```

然后将 train.txt 放在./scripts/文件夹下面，因为在 src/yolo.c 文件会引用 train.y=txt。

```
void train_yolo(char *cfgfile, char *weightfile)

{

    char *train_images = "path/to/scripts/train.txt";

    char *backup_directory = "/path/to/backup/"; # 用绝对路径

}
```

（5） 生成标签文件
打开./data/labels/make_labels.py
加入需要生成的标签，注意标签的文件名 stopsign.png 和 yeildsign.png 需要与存放图像的文件夹 images 和存放框信息的 labels 文件夹下面的文件夹名称相同。

对应在 darknet/src/yolo.c 中是：

```
void run_yolo(int argc, char **argv)

{

    int i;

    for(i = 0; i < 20; ++i){

        char buff[256];

        sprintf(buff, "data/labels/%s.png", voc_names[i]);
```

```
            voc_labels[i] = load_image_color(buff, 0, 0);
        }
    }
```

4 修改代码

需要修改的代码如下：

darknet/src/yolo.c
darknet/src/yolo_kernels.cu
darknet/cfg/yolo-tiny.cfg # 以 yolo-tiny 为例

yolo.c

```
# 修改路径

void train_yolo(char *cfgfile, char *weightfile)

{

    char *train_images = "path/to/scripts/train.txt";

    char *backup_directory = "/path/to/backup/";

    # backup_directory 用绝对路径，否则会出现一下错误：

    # Saving weights to /backup/yolo-tiny-2class_100.weight
s

    # Couldn't open file: /backup/yolo-tiny-2class_100.weig
hts

    srand(time(0));

    data_seed = time(0);

    char *base = basecfg(cfgfile);

    printf("%s\n", base);

    float avg_loss = -1;
```

```
    network net = parse_network_cfg(cfgfile);

    if(weightfile){

        load_weights(&net, weightfile);

    }

    printf("Learning Rate: %g, Momentum: %g, Decay: %g\n",
net.learning_rate, net.momentum, net.decay);

    int imgs = net.batch*net.subdivisions;

    int i = *net.seen/imgs;

    data train, buffer;




    layer l = net.layers[net.n - 1];


    int side = l.side;

    int classes = l.classes;

    float jitter = l.jitter;


    list *plist = get_paths(train_images);

    //int N = plist->size;

    char **paths = (char **)list_to_array(plist);


    load_args args = {0};

    args.w = net.w;

    args.h = net.h;

    args.paths = paths;
```

```
        args.n = imgs;
        args.m = plist->size;
        args.classes = classes;
        args.jitter = jitter;
        args.num_boxes = side;
        args.d = &buffer;
        args.type = REGION_DATA;

        pthread_t load_thread = load_data_in_thread(args);
        clock_t time;
        //while(i*imgs < N*120){
        while(get_current_batch(net) < net.max_batches){
            i += 1;
            time=clock();
            pthread_join(load_thread, 0);
            train = buffer;
            load_thread = load_data_in_thread(args);

            printf("Loaded: %lf seconds\n", sec(clock()-time));

            time=clock();
            float loss = train_network(net, train);
            if (avg_loss < 0) avg_loss = loss;
            avg_loss = avg_loss*.9 + loss*.1;
```

```
        printf("%d: %f, %f avg, %f rate, %lf seconds, %d ima
ges\n", i, loss, avg_loss, get_current_rate(net), sec(cloc
k()-time), i*imgs);

        if(i%1000==0 || (i < 1000 && i%100 == 0)){

            char buff[256];

            sprintf(buff, "%s/%s_%d.weights", backup_directo
ry, base, i);

            save_weights(net, buff);

        }

        free_data(train);

    }

    char buff[256];

    sprintf(buff, "%s/%s_final.weights", backup_directory,
 base);

    save_weights(net, buff);

}

# 对类别的修改

char *voc_names[] = {"stopsign", "yeildsign"};

image voc_labels[2];

......

void test_yolo(char *cfgfile, char *weightfile, char *file
name, float thresh)

{

    draw_detections(im, l.side*l.side*l.n, thresh, boxes,
probs, voc_names, voc_labels, 2);
```

```
}

......

void run_yolo(int argc, char **argv)

{

    int i;

    for(i = 0; i < 2; ++i){

        char buff[256];

        sprintf(buff, "data/labels/%s.png", voc_names[i]);

        voc_labels[i] = load_image_color(buff, 0, 0);

    }

}
```

yolo_kernels.cu

```
void *detect_in_thread(void *ptr)

{

    float nms = .4;



    detection_layer l = net.layers[net.n-1];

    float *X = det_s.data;

    float *predictions = network_predict(net, X);

    free_image(det_s);

    convert_yolo_detections(predictions, l.classes, l.n,
l.sqrt, l.side, 1, 1, demo_thresh, probs, boxes, 0);

    if (nms > 0) do_nms(boxes, probs, l.side*l.side*l.n, l.
classes, nms);

    printf("\033[2J");
```

```
    printf("\033[1;1H");

    printf("\nFPS:%.0f\n",fps);

    printf("Objects:\n\n");

    draw_detections(det, l.side*l.side*l.n, demo_thresh, b
oxes, probs, voc_names, voc_labels, 20); # 20->2

    return 0;

}
```

yolo-tiny.cfg

```
[connected]

output= 1470 # SxSx(Bx5+class_num)

activation=linear


[detection]

classes=20 # 改成实际的 class_num


coords=4 #框框的 4 个坐标


rescore=1 # 得分


side=7 # 分的越多，检测的可能越准

num=2

softmax=0

sqrt=1

jitter=.2
```

```
object_scale=1

noobject_scale=.5

class_scale=1

coord_scale=5
```

5 pre-train

yolo 中用到的 pre-trained weights 的格式是.conv.weights 的文件，根据不同的 model，要对已有的 weights 进行转换。

三种模型对应于不同的 weight
yolo.cfg -> extraction.conv.weights
yolo-small.cfg -> strided.conv.weights
yolo-tiny.cfg -> darknet.conv.weights

```
./darknet partial cfg/extraction.cfg path/to/extraction.we

ights extraction.conv.weights 25 # ./darknet partial 转化网

络 现有 weights 的路径 需要生成的 weights 的路径



./darknet partial cfg/darknet.cfg path/to/darknet.weights
path/to/darknet.conv.weights 14
```

6 training 使用 cpu 模式进行训练

```
$ make #需要 make 一下

$ ./darknet yolo train cfg/yolo-tiny.cfg path/to/darknet.conv.weig
hts
```

7 将生成的权重文件拷贝到 darknet_ros 包中

8 依次启动 kinect v2 驱动包， 启动 darknet_ros 包，启动主程序包 beginner_tutorials