Carnegie Mellon University

Individual Lab Report #2

Author:
Yuqing Qin

Team C: COBORG
Gerald D'Ascoli | Jonathan Lord-Fonda | Yuqing Qin | Husam Wadi
Feng Xiang

March 4, 2021

Table of Content

1 Individual Process

The Coborg platform is a wearable robotic arm that can help people hold the target objects overhead. My work in the Coborg project mainly focused on the Vision Subsystem design and implementation. In particular, I worked on target object localization tasks. To do so, our team came up with the hand detection idea to indicate the target object location when people hold the objects. During the last few weeks, I worked on the YOLO hand detection model setup and Realsense D435i depth camera setup. By the first progress review, I have already integrated the camera ROS node with the YOLO ROS node and evaluated the hand detection performance on GPU settings.

1.1 Depth Camera

To extract the 3D position of the target object, the camera currently using is the Intel Realsense D435i. This camera has an open-source built-in ROS package. Therefore, following the instructions on Github, I set up the camera and installed the camera ROS package. The depth camera contains RGB raw images, combined with depth images by default. When launching the ROS node, by changing input parameters, the camera can also publish the point cloud information. I also used Rviz to visualize the camera outputs, specifically the RGB raw image from the '/camera/color/image_raw' topic, the depth image from '/camera/depth/image_rect_raw', and point clouds from '/camera/depth/color/points'. Figure 1 shows the raw images demonstrations from the D435i camera.

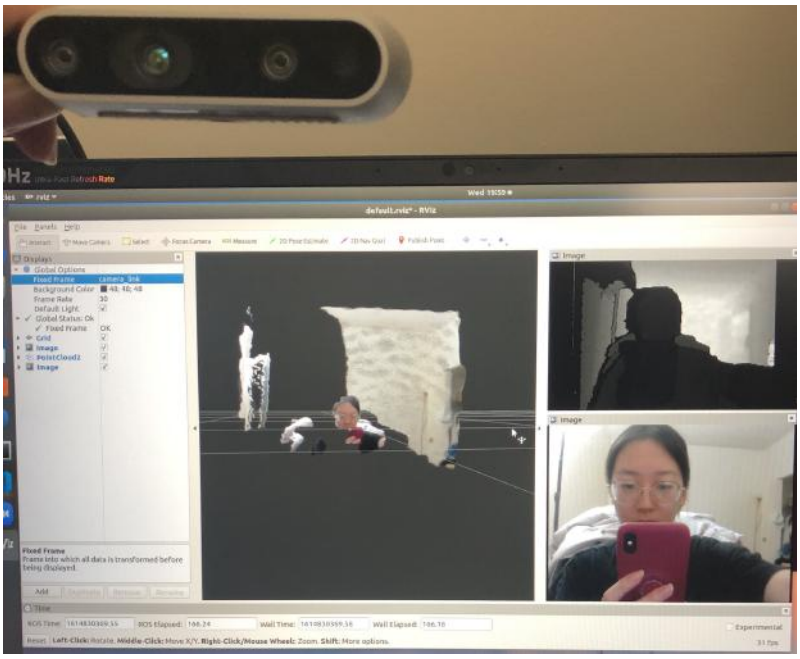

Figure 1. Depth camera demonstration (middle: pointcloud, upper right: depth image, bottom right: RGB raw image)

1.2 Hand Detection

Hand detection is the main task for our project. To implement it, I checked several hand detection algorithms, including YOLO, SSD, and OpenPose. By comparing their running performance, specifically accuracy and run time, we have decided to use YOLO v3. YOLO (You Only Look Once) is a real-time object detection algorithm that is commonly used in localization tasks. Even though the hand class is not covered by the original YOLO model, I found an open-source pre-trained hand detection model on YOLO v3, which is trained on several hand datasets and has relatively good accuracy. Therefore, there is no need for us to retrain the YOLO. By testing the pre-trained model, I found it ran slower on the CPU than I expected. The approximate run time is around 2 FPS, which will not fulfill our performance requirements. I also tested YOLO v3 tiny, which turned out to be a little bit better than YOLOv3, because the tiny model has a simpler structure compared to YOLO v3. Later, I set up GPU Cuda and CuDNN, which improved the runtime performance a lot. As a result, the YOLO v3 tiny can run with 200 FPS, and YOLOv3 can run with 30FPS right now.

Moreover, I launched the open-source YOLO ROS wrapper, which is called ‘darknet_ros’. This wrapper took YOLO pre-trained model weights and configurations as the input and published the bounding boxes into the ROS topic. By setting up the input camera topic and YOLO model configurations, I integrated the depth camera with the YOLO v3 hand detection model within the ROS wrapper.

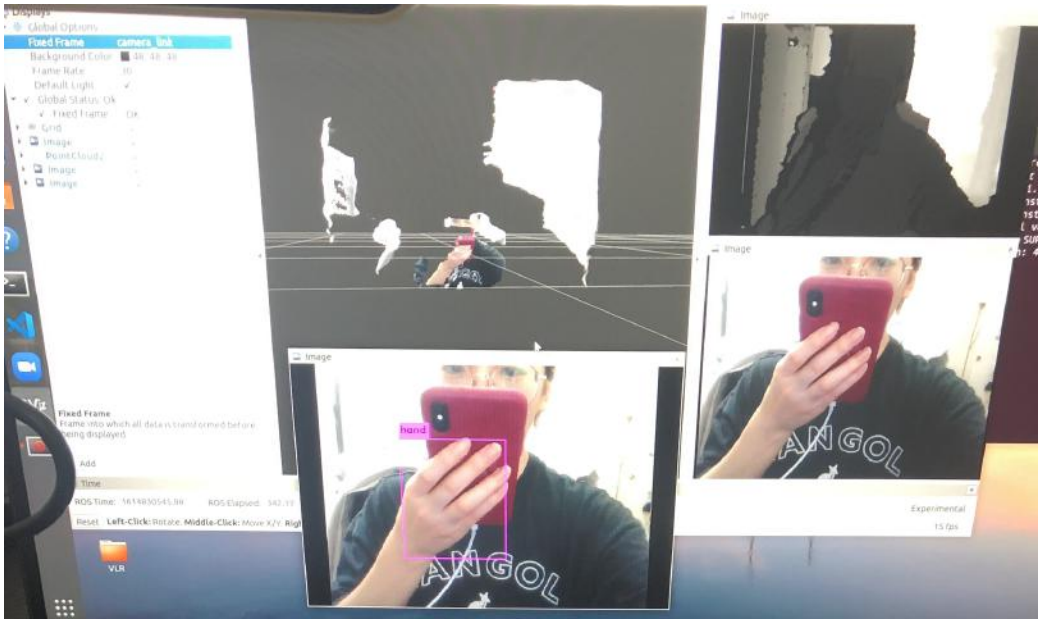

Figure 2. Hand detection demonstration (bottom: hand detection bounding box)

2 Challenges

The main challenge I faced for the past few weeks was the running speed for YOLO v3. We set up the performance requirement for the run time within 5 seconds, however, the YOLO v3 running on the CPU was slower than the requirement. I further tried different models, also tried YOLO v4 and YOLO v3 tiny. It turned out that the tiny version of YOLO is faster on CPU settings compared to others, but still not good enough. By setting up GPU and modifying the Makefile for YOLO, I improved the run time performance to 30FPS, which then is a real-time detection.

The current challenge I faced is outputting the corresponding 3D position of hands from the bounding box and depth information. It may also require the internal camera matrix (intrinsics) to work together. Therefore, I may need more research on how to accomplish this requirement and ensure the run time performance at the same time.

3 Teamwork

Team Member	**COBORG Progress**
Feng Xiang	-Created URDF model -Integrated URDF with Move-It -Got Robot Arm to Move to a point selected in RVIZ
Jonathan Lord-Fonda	-Updated ROS Node Map with Gains node -Updated ROS Node Map for vision system -Added a speaker output to ROS Node Map -Wrote Main State Node -Wrote up semantic Goal_Setter Node -Helped construct the robot holder/testing structure
Gerry D'Ascoli	-Redesigned voice recognition system to work with "Coborg" trigger word -Wrote script to translate verbal commands into robot instructions -Added audio feedback for voice control system
Husam Wadi	-Cleaned up B512 with the group -Built structure to hold Coborg -Adapted timeline to increased workload -Assisted Jason with intel realsense (T265) -Assisted Yuqing with intel realsense (D435i) -Assisted Jonathan with ROS Node Map -Ensured team used Github

Table 1. Teamwork for Coborg

4 Plans

For the next two weeks, I will first work on extracting the hands' depth information from the depth camera. Also, I will mainly focus on the ROS node (Goal_setter), which should align the bounding box information to the point cloud and depth to obtain the 3D position of detected hands. I will do some research and design work before starting to implement it. I also need to talk to Jonathan about the ROS node map design before implementation. Besides, I will work on the tracking camera (Intel Realsense T265) and set up ROS nodes for it.