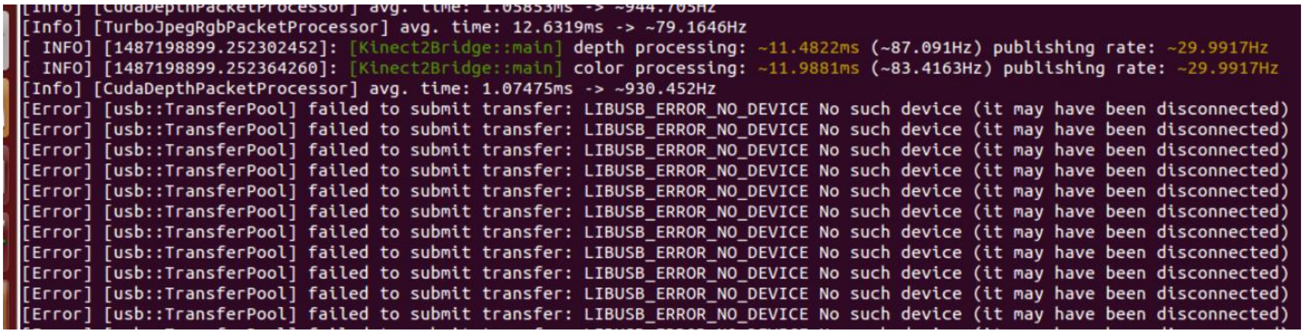
Che-Yen Lu
Team E: PLAID
Teammates: Michael Beck, Akshay Bhagat,
Matt Lauer, Jin Zhu
ILR07
February 16, 2017

1. Individual progress

For this progress review, we aim for integrating all sub-systems to demonstrate whole picking scenario. In demonstration, items are classified and bounding boxes are created by vision sub-system, and grasping sub-system will ask such information to grasp items based on their position. My individual progress for this goal includes three main parts: JSON parser, multiple Kinect installation and all system Integration.

JSON parser is a crucial function for Amazon competition. Bins' configurations and picking items are provided by JSON file before the competition, which means system need to parse JSON file first to know current items' locations beforehand, so I create a rosnode written by Python. The reason why I choose python is simple: JSON libraries in python are handy to use. There are three service advertised in this node, which are bin/tote content, bin/tote update, and output JSON, respectively. When system executed a successful/unsuccessful pick, items' locations will also be updated by using bin/tote update. Bin/tote content is used when system is initializing its environment, and output JSON is used when time is up or mission complete.

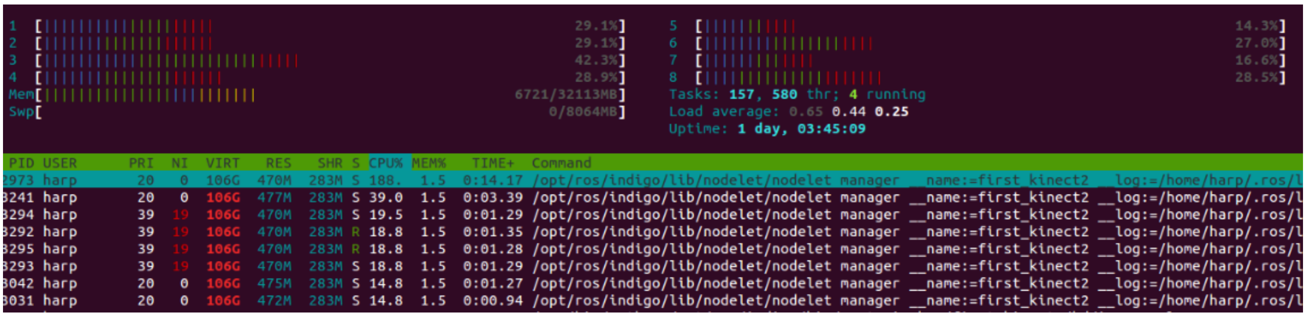
I spend most of my time on investigating how to enable multiple Kinect on one computer. However, USB bandwidth issue occur. I try to launch two Kinect by existing hardware, but one of Kinect can't even be launched if they share the same USB bus. This issue is mitigated by adding another USB PCI-E adapter since USB bus is not overloading all the time now. Second issue is about unstable connections. Although two Kinects are connected and functional sometimes, the connections are really unstable, and the USB error messages pop up for no reason. I can't conclude what the root cause is behind this now, but I believe USB extension cable is related to it. The error messages are shown in Figure 1.



```
[Info] [CudaDepthPacketProcessor] avg. time: 1.05853ms -> ~944.705Hz
[Info] [TurboJpegRgbPacketProcessor] avg. time: 12.6319ms -> ~79.1646Hz
[ INFO] [1487198899.252302452]: [Kinect2Bridge::main] depth processing: ~11.4822ms (~87.091Hz) publishing rate: ~29.9917Hz
[ INFO] [1487198899.252364260]: [Kinect2Bridge::main] color processing: ~11.9881ms (~83.4163Hz) publishing rate: ~29.9917Hz
[Info] [CudaDepthPacketProcessor] avg. time: 1.07475ms -> ~930.452Hz
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
[Error] [usb::TransferPool] failed to submit transfer: LIBUSB_ERROR_NO_DEVICE No such device (it may have been disconnected)
```

Figure 1. Kinect USB error messages

Multiple Kinects can't be turned on at the same time because the IR camera could interfere with each other. Also, the CPU loading is really high if HD RGB images and SD point cloud are subscribed. Figure 2 shows the CPU loading by only activating one Kinect. It consumes almost two CPUs! Since Kinect ROS package is a CPU monster, the realistic way to tackle such problem is to turn Kinect on/off on demand. In ROS, shutdown and subscribe are corresponding APIs.



```
  1  [||||||||||||||||||||                     29.1%]     5  [|||||||||||                             14.3%]
  2  [||||||||||||||||||||                     29.1%]     6  [||||||||||||||||||||                    27.0%]
  3  [||||||||||||||||||||||||||||||           42.3%]     7  [||||||||||||                            16.6%]
  4  [|||||||||||||||||||||                    28.9%]     8  [|||||||||||||||||||||                   28.5%]
  Mem[|||||||||||||||||||||||||          6721/32113MB]     Tasks: 157, 580 thr; 4 running
  Swp[                                       0/8064MB]     Load average: 0.65 0.44 0.25
                                                           Uptime: 1 day, 03:45:09

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
2973 harp       20   0  106G  470M  283M S 188.  1.5  0:14.17 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
3241 harp       20   0  106G  477M  283M S 39.0  1.5  0:03.39 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
3294 harp       39  19  106G  470M  283M S 19.5  1.5  0:01.29 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
3292 harp       39  19  106G  470M  283M R 18.8  1.5  0:01.35 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
3295 harp       39  19  106G  470M  283M R 18.8  1.5  0:01.28 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
3293 harp       39  19  106G  470M  283M S 18.8  1.5  0:01.29 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
3042 harp       20   0  106G  475M  283M S 14.8  1.5  0:01.27 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
3031 harp       20   0  106G  472M  283M S 14.8  1.5  0:00.94 /opt/ros/indigo/lib/nodelet/nodelet manager __name:=first_kinect2 __log:=/home/harp/.ros/l
```

Figure 2. CPU loading when Kinect is activated

Integration is also one of the most important task for this PR. I integrate the Kinect interface with fast R-CNN first and integrate all vision system with grasping sub-system after. The classification and bounding box produced by fast R-CNN are accurate if items are not occluded by other items. Figure 3 shows the result of fast R-CNN.

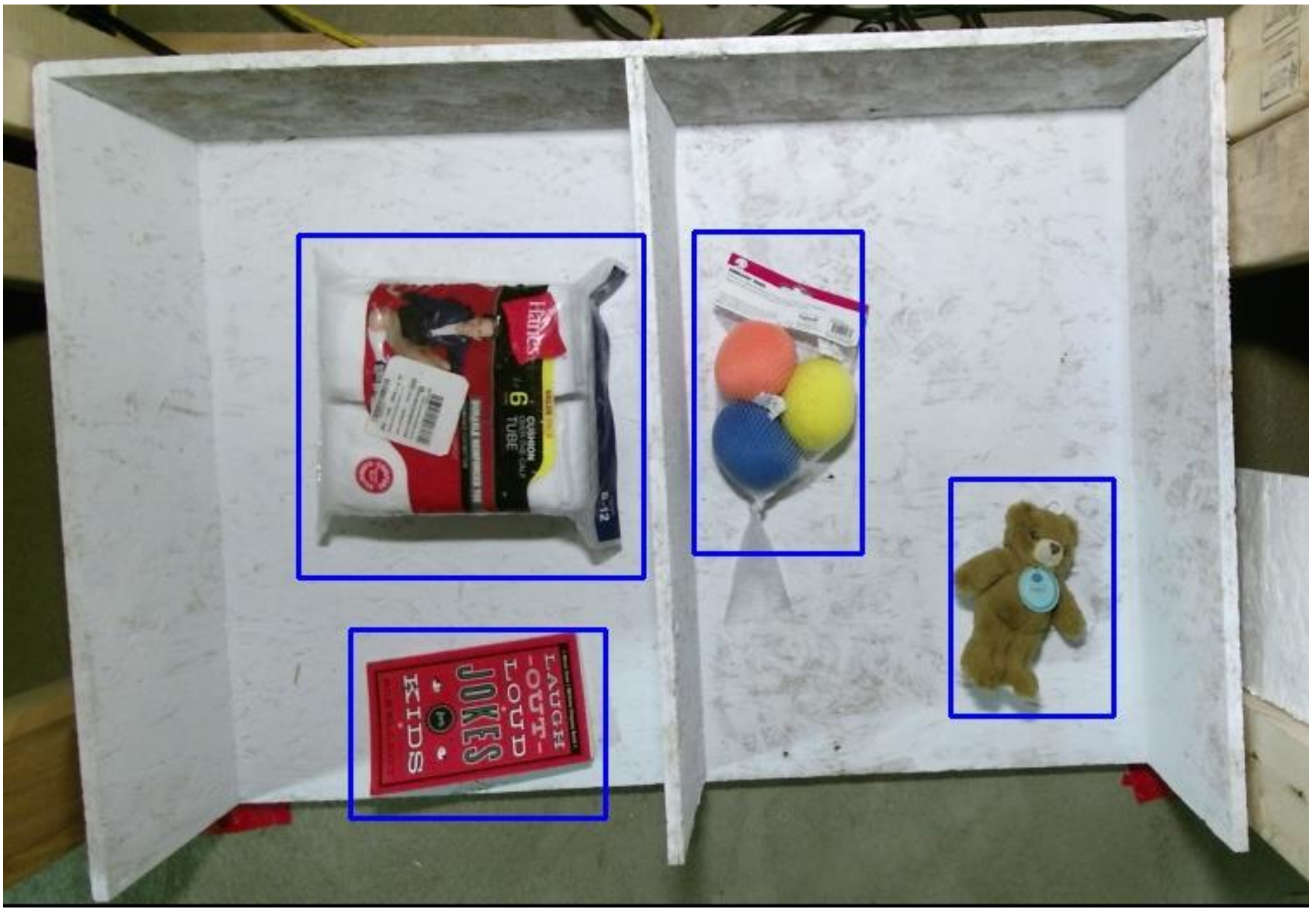


Figure 3. Bounding box produced by fast R-CNN

2. Challenges

As far as I can see, the biggest risk now is the Kinect stability issues. When the Kinect fails, the only way to recover from it is to relaunch the launch file. One of the possible solution is to think a detection mechanism to detect how long the Kinect is being unresponsive, like watchdog program.

If the failure is detected, watchdog will try to kill Kinect process and relaunch it. However, I don't think such workaround is easy to apply to our system. First, the state machine will be really unpredictable since Kinect could crash at any moment. Second, it is possible the Kinect still can't be detected even we relaunch it, which means it's hardware issue and unplug USB is needed.

Although we want to keep code clean and neat, I don't think we could still keep this promise when the schedule is tight. Usually, we only reserve one day to integrate all things, and the coding style and modularity don't matter at all when things don't work. In ideal world, test and debugging should occupy above 50% of the developing time since quality code could save us lots of time. However, code quality takes time and effort, I want team to have consensus on this. I will bring this up this week.

3. Teamwork

For progress review seven, we focus on different domains and break down the tasks as follows:

- Michael Beck – Project manager. Michael handles project schedule and goal. He keeps helping team to break tasks down and monitor progress of sub-tasks.
- Akshay Bhagat – Akshay helped team to implement Fast-RCNN python wrapper. Fast RCNN could identify 5 2016 APC items now.
- Matt Lauer – Matt create planning scene for linear actuator and UR10. He also implemented simple top-down grasping strategies.
- Che-Yen Lu – I create a JSON parser, which is a rosnode. I get two Kinects working although they are not stable at all. I also help team to integrate software, includes state machine, vision, and grasping sub-systems.
- Jin Zu – Jin helps MSCV teammate to annotate images.

4. Future plans

For my personal plan, I may move to grasping sub-system to implement grasping strategies. Deformable objects may be the first priority now for sure.

For team, we target at finishing fully functional frame before the middle of April. As for perception, several CNN methods will be evaluated under different environments and scenarios. Mechanism for camera calibration will also be designed and implemented before PERCH comes into play. Schedule of unknown objects' identification will be pushed off till we are confident with the result of known items.