ON MY DEVICE (ricky), THE PASSWORD IS ( password )

Devices used: Jetson Nano Dev Kit (64GB microSD card), RealSense D435i camera

First, flash the image card and setup per <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit>

<https://github.com/IntelRealSense/realsense-ros> NOTE: THERE IS A “JETSON INSTALLATION GUIDE” LINK THERE

You will need at least 3 terminals in this order:

roscore

roslaunch realsense2\_camera rs\_camera.launch enable\_infra1:=true enable\_infra2:=true initial\_reset:=true enable\_accel:=true enable\_gyro:=true

rviz

Then you can add an Image in ROS and subscribe to which camera you want

Fun option: To save power (keyboard, mouse, display), you can SSH (SoftSHell) into the Jetson Nano through your internet connection. To do this, I used PuTTy and Xming on MY PERSONAL COMPUTER. Tip: Save your settings so you can load in with X11 forwarding into the Jetson IP easily in the future!

<https://cat.pdx.edu/platforms/windows/remote-access/xming/>

You will need to find the Jetson’s IP address (ip addr show)

YOU WILL WANT TO OPEN “XLAUNCH”, NOT “XMING” 😊

SLAM example installation can be found [here](https://github.com/IntelRealSense/realsense-ros/wiki/SLAM-with-D435i), but replace “kinetic” with “melodic”

I wasn’t able to get the map to update, but running the launcher as shown below (with initial\_reset=true) helped mitigate the global fixed frame error

ROS melodic commands:

RVIZ:

Terminal 1:

Roscore

Terminal 2:

roslaunch realsense2\_camera rs\_camera.launch enable\_infra1:=true enable\_infra2:=true initial\_reset:=true enable\_gyro:=true enable\_accel:=true unite\_imu\_method:=linear\_interpolation

Terminal 3:

rviz

SLAM

Terminal 1:

Roscore

Terminal 2:

roslaunch realsense2\_camera opensource\_tracking.launch

You can try some of the add-ons from RVIZ terminal 2 above

If you can’t see any options from the camera you may need to do the rs\_camera.launch above

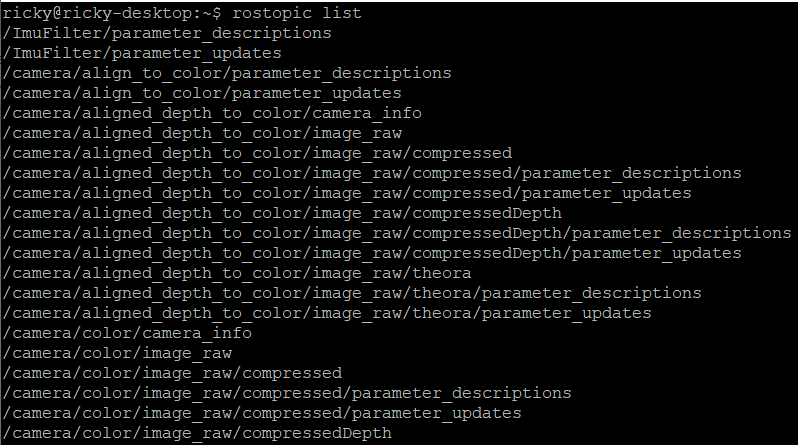
Note: At some point after catkin\_make, I did not need to open roscore anymore (I think it got implemented into the opensource\_tracking.launch)

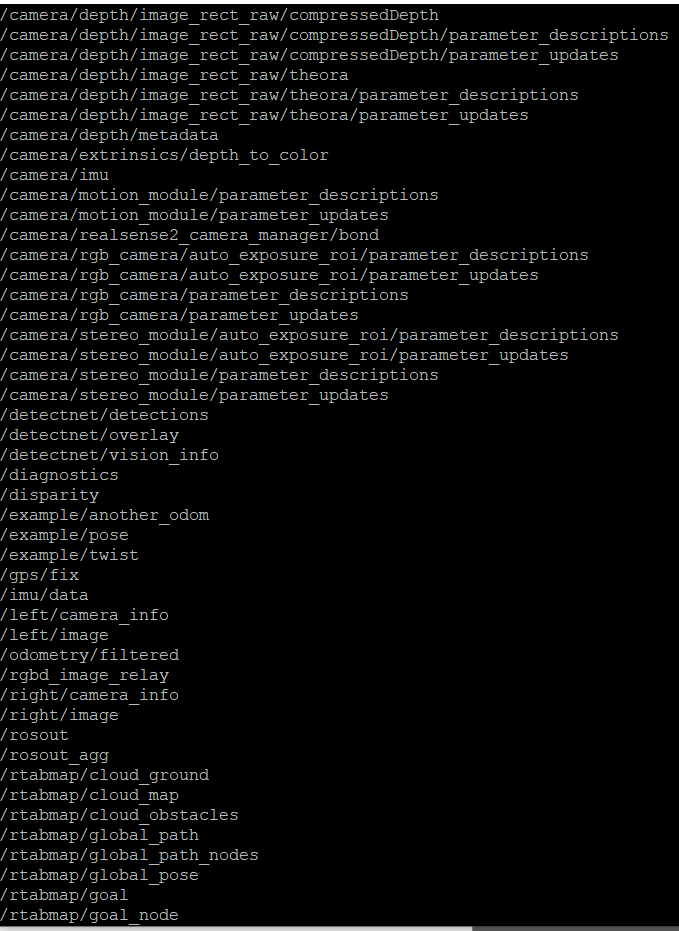
nodes available with opensource tracking:

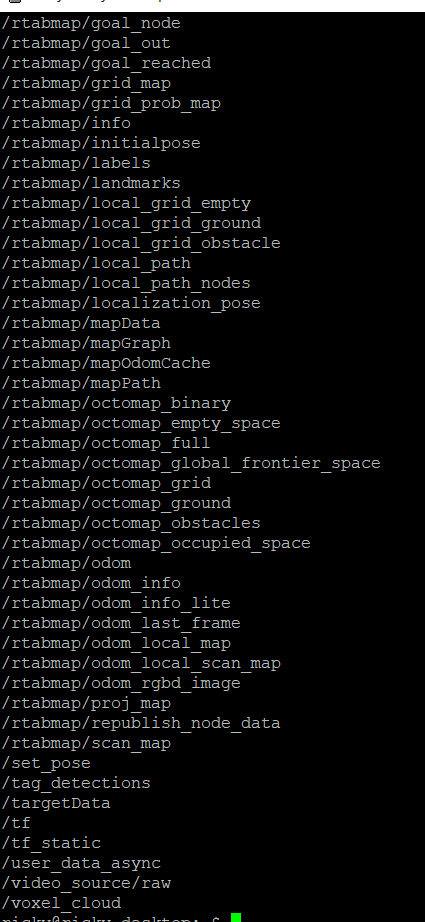
imu/data

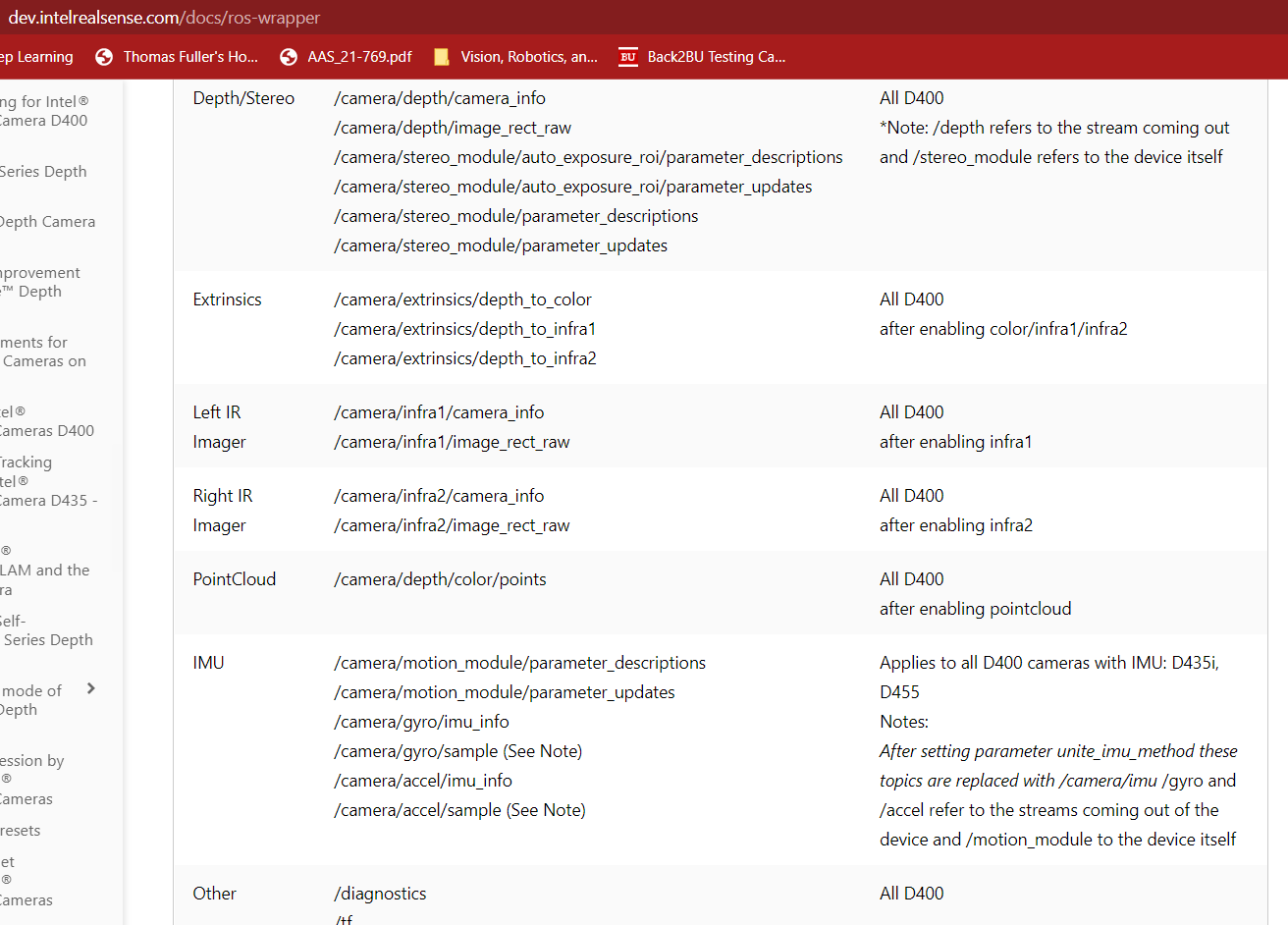
and

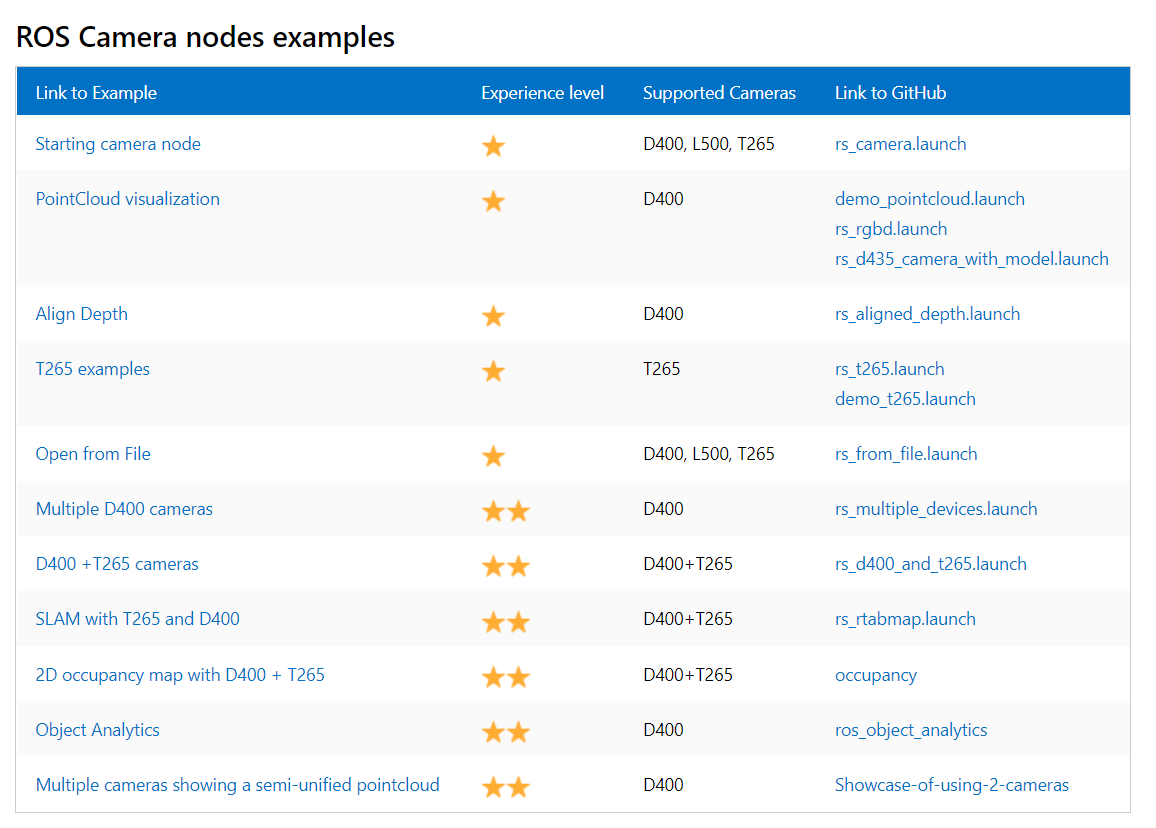
camera/imu

seem optimistic  










To create your own map in rviz (I didn’t do much with this):

rosrun tf static\_transform\_publisher 0 0 0 0 0 0 1 map my\_frame 10  
  
roslaunch realsense2\_camera rs\_camera.launch enable\_infra1:=true enable\_infra2:=true  
  
roslaunch realsense2\_camera rs\_camera.launch unite\_imu\_method:=linear\_interpolation enable\_infra1:=true enable\_infra2:=true align\_depth:=true

Once I catkin\_make to create ricky\_ws, this only works with the ricky\_ws source devel/setup.bash!!  
  
roslaunch realsense2\_camera rs\_camera.launch filters:=pointcloud  
  
DEEP LEARNING ROS:  
REMEMBER TO SOURCE DEVEL/SETUP.BASH!  
Video viewer:  
roslaunch ros\_deep\_learning video\_viewer.ros1.launch input:=/dev/video2 output:=display://0

Imagenet classifier:  
roslaunch ros\_deep\_learning imagenet.ros1.launch input:=/dev/video2 output:=display://0

Detectnet (fun!):

roslaunch ros\_deep\_learning detectnet.ros1.launch input:=/dev/video2 output:=display://0

remember, the above needs to be cd catkin\_ws and source devel/setup.bash

rosrun beginner\_tutorials talker\_RH.py

but the python scripts in a new terminal need to be cd ricky\_ws and source devel/setup.bash

by the way if you ever want to save a rostopic output to a .txt file, run

rosrun beginner\_tutorials listener\_RH.py

I rigged it to make data collection + parsing easier.

Unfortunately camera data made a huge text file? So I installed glogg to try to open it?

Now if you want RealSense D435i nodes AND the detectnet nodes, the camera launch file doesn’t play well with the detectnet launch file. However, the SLAM launch files opens up some nodes to share from the D435i camera:

roslaunch realsense2\_camera opensource\_tracking.launch

I could not get X11 forwarding via SSH to work with these, so I had to plug in mouse + keyboard to the Nano.

**Remember, video0 is error but I think it should be depth, video1 is infrared and video2 is RGB**

**Interesting command:**

ls /dev/video\* -1a | xargs -n1 v4l2-ctl --all -d

ROS 2:

The above stuff was done with ROS1 (ros-melodic-desktop)

Google says that ROS1 and ROS2 can coexist, but you should change your setup.bash file if you want to switch between them. Else, run the command below in EVERY terminal:

source /opt/ros/dashing/setup.bash

You could also just edit the bashrc file, near the end of it, with

gedit ~/.bashrc

I followed the instructions here to install and test ROS2. Note since the Jetson Nano had Ubuntu 18.04 Bionic (instead of 20.04 Focal), I believe an end-of-life ROS2 distribution, ROS Dashing, had to be installed.

<https://docs.ros.org/en/dashing/Installation/Ubuntu-Install-Debians.html>

This also talks about the ROS1 bridge. I was not able to get very far with attempts at that.

OPENCV: For rtabmap and other applications, I needed to reinstall a later version of OpenCV on the Nano (note this is different from the python package “opencv”, which you can download from terminal with “pips install opencv”)

<https://github.com/opencv/opencv.git>

plus

sudo apt-get install python3-opencv

although I only had it on python (which is python2), not on python3, I’m not sure what that is about

Useful Linux commands:

gnome-system-monitor

(system monitor for memory usage)

eog <filename>

(displays image; useful on SSH)

gedit

(text editor)

vim

(alternative text editor, may need to be installed)

Machine learning projects:

<https://www.youtube.com/watch?v=bcM5AQSAzUY> (Object detection)

I actually installed this one on mine. I installed the default models as shown in the video, and also installed Pytorch.

It has image databases and pre-trained image classification models to use.

It also installs Pytorch, which I used in my Deep Learning class for machine learning (training and testing models). I think this is where you would want to focus efforts on identifying specific targets for the Nano to recognize.

The video also walks through creating your own Python script on the Nano, which will definitely be useful.

I could not get X11 forwarding to work (OpenGL issue… Hoping to fix later). To launch the image classification Python script, go to “cd jetson-inference” and type

Python my-detection.py

Note that mobilenet-v2 seems to be the only pre-loaded model. There are some that have added their own experimental models in forums, and with Pytorch you could train your own. I am not sure what steps that would take.

<https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python> (connect RealSense D435i camera to Python scripts)

Python on Nano for dummies:

<https://automaticaddison.com/how-to-write-a-python-program-for-nvidia-jetson-nano/>

tip: to pip install python3 packages, such as opencv, onto the Jetson, type this into terminal:

pip3 install opencv-python

I needed to first pip install scikit to get skbuild, then

pip3 install --upgrade pip setuptools wheel

then update protobuf with

pip3 install –upgrade protobuf

Additional hardware concerns/ideas:

According to this wiki, these other cameras are a little more plug-and-play with the Nano (less setup etc needed): <https://elinux.org/Jetson_Nano#Cameras>

This cheap part supposedly adds wifi and Bluetooth capabilities:

<https://learn.sparkfun.com/tutorials/adding-wifi-to-the-nvidia-jetson/all>

If someone has an existing system they would like to implement into ROS, this tutorial seems to describe how to create a ROS Wrapper with Python/C++:  
<https://roboticsbackend.com/create-a-ros-driver-package-introduction-what-is-a-ros-wrapper-1-4/>

Saving Data in ROS environment:

In ROS you save data to rosbag (.bag) files

Here is one source for converting to csv using a custom python script:

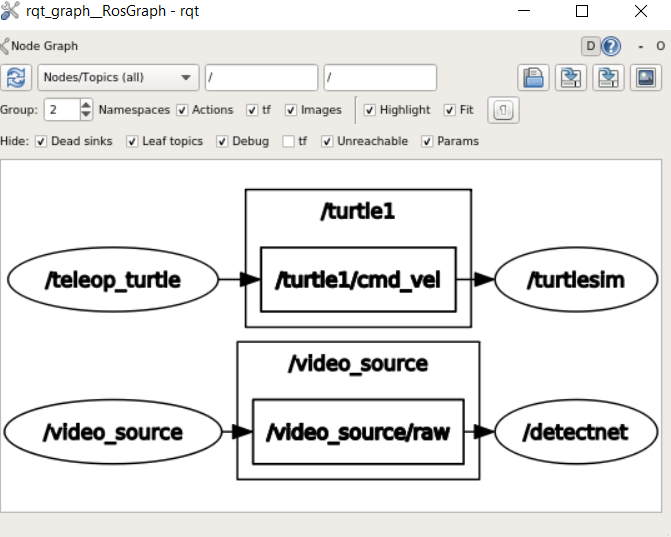
<https://www.youtube.com/watch?v=AnaLC0A4g7E>

I put this in my ricky\_ws. Nifty manual labor, but since it is user interface I’d rather figure out making my own ROS publisher or Python script.

1. Roscore
2. rosrun rosbag\_to\_csv rosbag\_to\_csv.py

Creating a ROS package tutorial: <http://wiki.ros.org/ROS/Tutorials/CreatingPackage>

I kept following the subsequent tutorials to try to learn how to make my own publisher later.



http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

**Getting Python into your ROS package** (when you catkin\_make with your package /src/<your package here> directory (for example the beginner\_tutorials that I did with the python listener and python talker); and that directory will have a CMakeLists.txt file (NOT the master file in your catkin workspace, but for that specific package). Editing this file looked tricky but the tutorial gave a well-documented default file to edit.

I used gedit as a text editor. I would, for example, go to ricky\_ws/beginner\_tutorials/src/scripts and say

gedit talker.py

After saving the text file I would have to run another command so in the Linux terminal when I “ls” the directory, the .py files would show up green

chmod +x setup.py

I did need to copy+paste the setup.py file from online.

<http://docs.ros.org/en/kinetic/api/catkin/html/howto/format2/installing_python.html>

I think this will be useful:

<https://answers.ros.org/question/301607/how-to-use-data-of-subscriber-to-publish-and-command-the-robot/>

from std\_msgs.msg import String

from vision\_msgs.msg import Detection2DArray

from vision\_msgs.msg import Detection2D

def callback(msg):

#mydata = msg.detections[0].results[0]

#rospy.loginfo(msg.detections[0].results[0])

mydata=msg.detections[0]

myid = mydata.results[0].id #stop sign id is 13 in the label list

myscore=mydata.results[0].score

#mostly interested in id (detection) and score (confidence that the object is correctly labeled)

myboxox = mydata.bbox.center.x

myboxoy = mydata.bbox.center.y

print('id, confidence, x, y =',myid, myscore, myboxox, myboxoy)

#rospy.loginfo(mydata)

rospy.sleep(1) #pause

#I reverse-engineered the data indexing through trial-and-error and a lot of luck

def listener():

# In ROS, nodes are uniquely named. If two nodes with the same

# name are launched, the previous one is kicked off. The

# anonymous=True flag means that rospy will choose a unique

# name for our 'listener' node so that multiple listeners can

# run simultaneously.

rospy.init\_node('listenerRH', anonymous=True)

rospy.Subscriber('detectnet/detections', Detection2DArray, callback)

#will likely want this to be listening to detectnet/detections

# spin() simply keeps python from exiting until this node is stopped

rospy.spin()

SHELL SCRIPT!

I got my own shell (.sh) script, ricky\_script.sh , to work! It opens two additional terminals in the background to roslaunch the detectnet demo in one terminal, rosrun the Python messenger script in another terminal, then rostopic echo the messenger’s broadcast in the main terminal.

When running this, you can press ctrl+d to shut down all terminals once you are done observing.

If you open the script, I have some comments in there for how it works. It was actually simple once the syntax worked! just remember to chmod +x the file so it is executable.

Ricky Notes:

Jetson, Tom Fuller 4/14:

Code works off of Waypoints

string of (x,y,z) coordinates (position, OR VELOCITY!) in Python list [currently keeping set to constant z, i.e. 3m]

PositionTarget() is info sent over from ROS on the Pi

ROS's role is to relay messages between Raspberry Pi and IMU; may change soon with IMU updates

UNITS ARE IN METERS!

My script would determine location of tank (assume shoot straight down and we are above it)

Would need to tell which way to move and feed it to their existing script. Would change setpoint position (line 159 jk in lines 177-179, in a\_waypoint\_tester.py or w.e.)

detecting hazards is one thing they have done (used magnetometer and sensor data, find areas with magnetic field, decided it would indicate the location of a landmine)

One goal was landing a quadcopter on the ASV (see a landing pad, controlled landing, see a static landing position)

For now, pick say stop sign, find an object and know where it is relative to local coordinate frame of the quad to the object, track a "landing location"

DB:

catkin\_ws --> realsense dir --> new terminal roslaunch command --> new terminal and rostopic list --> how to save ROS topic is what DB is working now

rostopic echo the IMU data (make sure it's publishing) --> rosbag record rosout <topic name>

should save to a bag file

ctrl+c will stop saving and will create a .bag file

should be able to put it into a csv file one way or another (how to parse the data)

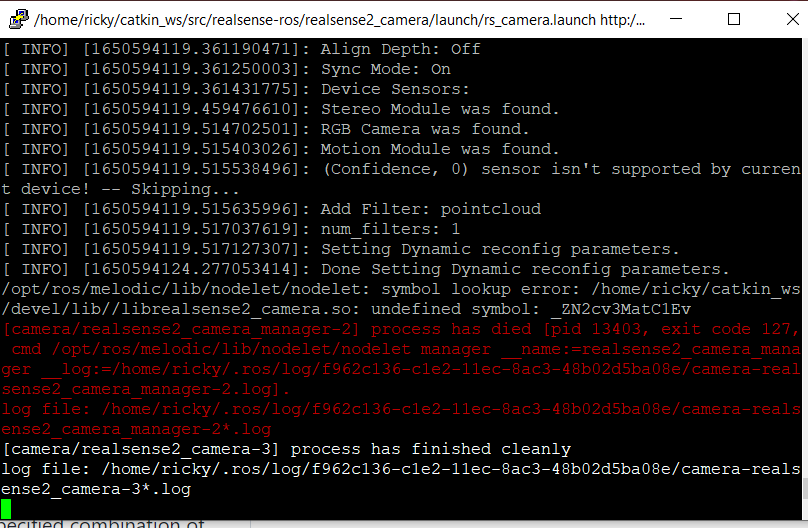
Ricky notes 4/21/2022

I was investigating how to pull IMU data in ROS through the SLAM and detectnet demos. I got them working in parallel, giving me a wide range of rostopics to look into.

Unfortunately the realsense camera launcher stopped working shortly after (some realsense camera manager has died error).

So the detectnet demo is working fine and tracking images.  
However, I am not sure I can get much else done with the depth camera.  
I will note that I have never had a reliable experience with the IMU, and I was more optimistic that the depth camera could give me some useful data. (I did subscribe to the depth cam rostopic and see raw image values earlier in the night).

Hopefully the camera sorts itself out. It may need a fresh install though. I tried to do sudo apt get update, and update the librealsense and realsense-ros packages



Detectnet labels list:

1. unlabeled
2. person
3. bicycle
4. car
5. motorcycle
6. airplane
7. bus
8. train
9. truck
10. boat
11. traffic light
12. fire hydrant
13. street sign
14. stop sign
15. parking meter
16. bench
17. bird
18. cat
19. dog
20. horse
21. sheep
22. cow
23. elephant
24. bear
25. zebra
26. giraffe
27. hat
28. backpack
29. umbrella
30. shoe
31. eye glasses
32. handbag
33. tie
34. suitcase
35. frisbee
36. skis
37. snowboard
38. sports ball
39. kite
40. baseball bat
41. baseball glove
42. skateboard
43. surfboard
44. tennis racket
45. bottle
46. plate
47. wine glass
48. cup
49. fork
50. knife
51. spoon
52. bowl
53. banana
54. apple
55. sandwich
56. orange
57. broccoli
58. carrot
59. hot dog
60. pizza
61. donut
62. cake
63. chair
64. couch
65. potted plant
66. bed
67. mirror
68. dining table
69. window
70. desk
71. toilet
72. door
73. tv
74. laptop
75. mouse
76. remote
77. keyboard
78. cell phone
79. microwave
80. oven
81. toaster
82. sink
83. refrigerator
84. blender
85. book
86. clock
87. vase
88. scissors
89. teddy bear
90. hair drier
91. toothbrush