Goal:
- To obtain the pose estimation of the heatpipes on the ground, based on the given bag file data. The bag file data currently consists of only point cloud & RGB image data. Identifying the heat pipes would be the first task. Pipe detection has to be reliable and accurate since there is a very small threshold of error for the robot to get on to the pipes.

Pose estimation can be carried out using either the point cloud data or rgb image. We can approach this problem using the point cloud library (PCL) and also with a neural network based detection and pose estimation. Once we have 2 approaches, we can compare the approaches and identify the best suited method to implement for the next phase.

Hardware:
- realsense d455 (first bag)
- Zed2i (second bag) 2023-03-07

Inputs:
- Point cloud
- Stereo Images
- Field of view Horizontal 87deg , vertical 58 deg

System Usage:
- ROS2 Humble
- Ubuntu 22.04
- GPU MAC M1/ GPU from Google Colab

Tasks
Week 1
- Setup ROS 2 environment
- Converted bag to a ROS 2 Format and did bag review to decide next steps
    - rosbags-convert ../single_depth_color_640x480.bag
    - Using repo rosbags.
    - Steps from readme:
    - git clone https://gitlab.com/ternaris/rosbags.git
    -
    - cd rosbags
    - python -m venv venv
    - . venv/bin/activate
    -
    - pip install -r requirements-dev.txt
    - pip install -e .
    -
    -
- Suggested using QR code but can’t do that
- Converted Stereo Images to .JPEG files (1000+) files after writing a rosnode
- Installed labelImg (Python package that has gui to label images)
- Downloaded and built RTABMAP and YOLOv7 code.

Week 2: 06/02/2023-10/02/2023

- Manual Image labeling and segregation for train and validation
- Figured out how to use GPU in VM. (not possible on m1 mac)
- Tried using mac and gpu but yolov7 doesn't support MPS M1 Gpu's yet. I tried updating code but that would have caused bugs and waste time.
- I then setup a Google Colab notebook to run the training model since it provides GPUs.
- The labeling was wrong due to prior labels in the package I used. So, I manually corrected the labels.
- Finally I updated the configs and ran the training. The weights will be stored in the google colab.
- Got the trained weights: `yolov7-custom16/weights/best.pt`
- Detection failed. Need to re-train with a better and cleaner dataset.
- **val:** `Scanning 'data/validate/labels.cache' images and labels... 13 found, 0 missing, 0 empty, 0 corrupted: 100% 13/13 [00:00<?, ?it/s]`
- `Class Images Labels P R mAP@.5 mAP@.5:.95: 100% 1/1 [00:02<00:00, 2.53s/it]`
- `all 13 16 0.00308 0.75 0.0033 0.000627`
- `Speed: 47.6/7.7/55.3 ms inference/NMS/total per 640x640 image at batch-size 32`
- Re trained the model on less data. And the results were still not good.

```
Namespace(augment=False, batch_size=32, conf_thres=0.001, data='data/custom_data.yaml', device='0', exist_ok=False, img_size=1280,
YOLOR 🚀 v0.1-121-g2fdc7f1 torch 1.13.1+cu116 CUDA:0 (Tesla T4, 15109.875MB)

Fusing layers...
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
IDetect.fuse
Model Summary: 314 layers, 36481772 parameters, 6194944 gradients
 Convert model to Traced-model...
 traced_script_module saved!
 model is traced!

/usr/local/lib/python3.8/dist-packages/torch/functional.py:504: UserWarning: torch.meshgrid: in an upcoming release, it will be req
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
val: Scanning 'data/validate/labels.cache' images and labels... 13 found, 0 missing, 0 empty, 0 corrupted: 100% 13/13 [00:00<?, ?it
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:03<00:00,  3.26s/it]
                 all          13          16     0.00205         0.5     0.00116     0.00017
Speed: 37.7/4.4/42.1 ms inference/NMS/total per 1280x1280 image at batch-size 32
Results saved to runs/test/data/validte/images2
```

- Taking a step back and doing more research on how to create the dataset properly.
- Investigated Yolo V3, RetinaNet, Faster R CNN, Mask R CNN, Cascade R CNN.

- Trained a yolo v3 model and got the weights. Got 99% in mAP@0.5 and prediction works.

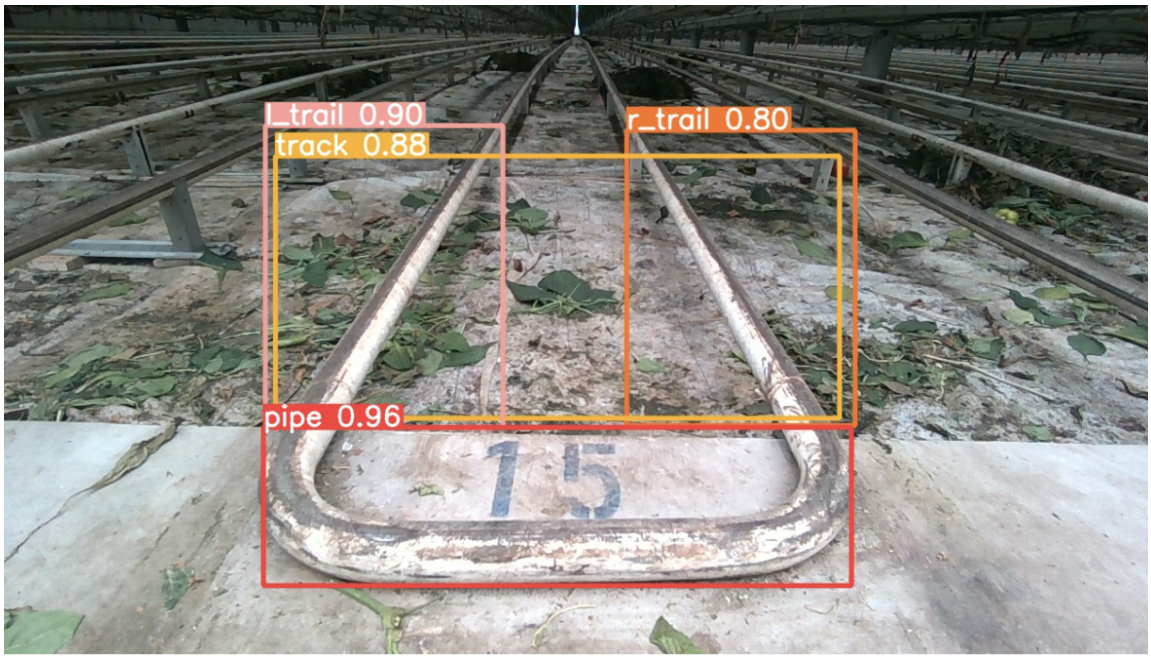

```
300 epochs completed in 0.414 hours.
Optimizer stripped from runs/train/exp12/weights/last.pt, 123.8MB
Optimizer stripped from runs/train/exp12/weights/best.pt, 123.8MB

Validating runs/train/exp12/weights/best.pt...
Fusing layers...
Model Summary: 261 layers, 61497430 parameters, 0 gradients
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  5.03it/s]
                 all         13         16      0.941          1      0.991      0.627
Results saved to runs/train/exp12
```

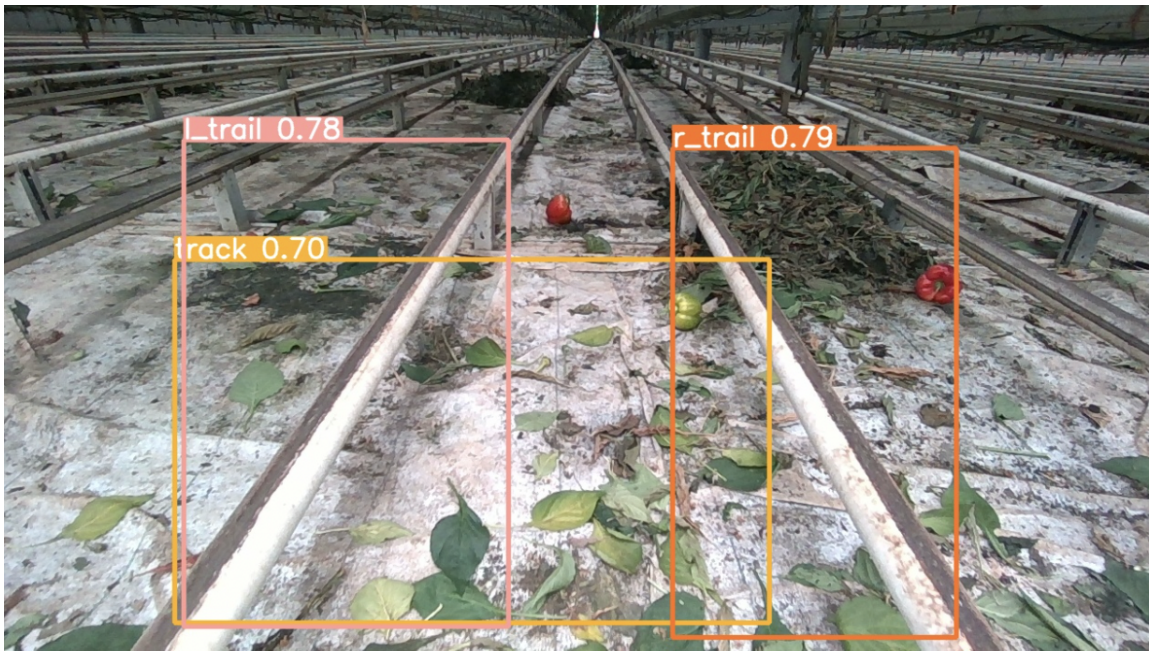
- Downloaded the weights. Now need to run it locally on all the images and find a way to run it when playing rosbag. (WIP)

Week 3 : 11/02/2023- 17/02/2023

- Goal: Detect horizontal trails of the pipe as well and do pose estimation using camera link and base link.
- Labeled images with left, right edges and track with pipe. (total 193 images)
- Train 162 images, validate: 31 images
- Weight stored in exp 14

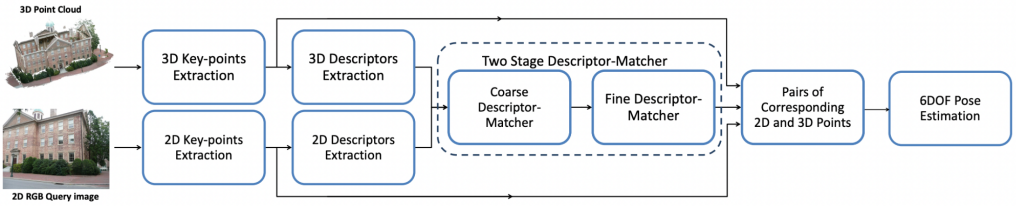
```
     296/299     38.7G    0.0157   0.05623  0.003241        87      1280: 100% 11/11 [00:06<00:00,  1.78it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.64it/s]
                   all         31        120      0.989      0.956      0.975      0.723

       Epoch   gpu_mem       box       obj       cls    labels  img_size
     297/299     38.7G   0.01609    0.0534  0.003251        90      1280: 100% 11/11 [00:06<00:00,  1.79it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.66it/s]
                   all         31        120      0.976      0.946      0.963      0.713

       Epoch   gpu_mem       box       obj       cls    labels  img_size
     298/299     38.7G   0.01591   0.05378  0.003269        76      1280: 100% 11/11 [00:06<00:00,  1.79it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.38it/s]
                   all         31        120      0.973       0.95      0.965      0.702

       Epoch   gpu_mem       box       obj       cls    labels  img_size
     299/299     38.7G   0.01627   0.05567  0.004082        68      1280: 100% 11/11 [00:06<00:00,  1.79it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.67it/s]
                   all         31        120      0.986      0.956      0.975       0.73

300 epochs completed in 0.711 hours.
Optimizer stripped from runs/train/exp14/weights/last.pt, 123.8MB
Optimizer stripped from runs/train/exp14/weights/best.pt, 123.8MB

Validating runs/train/exp14/weights/best.pt...
Fusing layers...
Model Summary: 261 layers, 61513585 parameters, 0 gradients
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.14it/s]
                   all         31        120      0.986      0.956      0.975       0.73
                  pipe         31         30          1      0.857      0.947      0.722
               l_trail         31         30      0.991          1      0.995      0.728
               r_trail         31         30      0.988          1      0.995      0.768
                 track         31         30      0.967      0.967      0.962      0.704
Results saved to runs/train/exp14
```

- 

- 

-
- For pose estimation, we have bounding boxes of left trail, right trail as well as track and pipe entry. We need to estimate pose of the bounding box/diagonal of box for trails w.r.t. Camera.
- Pose estimation libraries:
    - https://github.com/GeorgeDu/6d-object-pose-estimation#category-level
    - https://arxiv.org/pdf/2001.10609.pdf
    - https://github.com/topics/6d-pose-estimation
    - https://github.com/zubair-irshad/CenterSnap

- Learning techniques for pose estimation are trained on LineMOD dataset.The datasets include 3D object models and training and test RGB-D images annotated with ground-truth 6D object poses and intrinsic camera parameters.
- https://arxiv.org/pdf/2005.14502.pdf Use 3d point cloud and rgb image to extract key points, train the descriptors and create pair correspondence between 2d and 3d points to get 6 DOF Pose on new rgb images. (no open source code afaik)



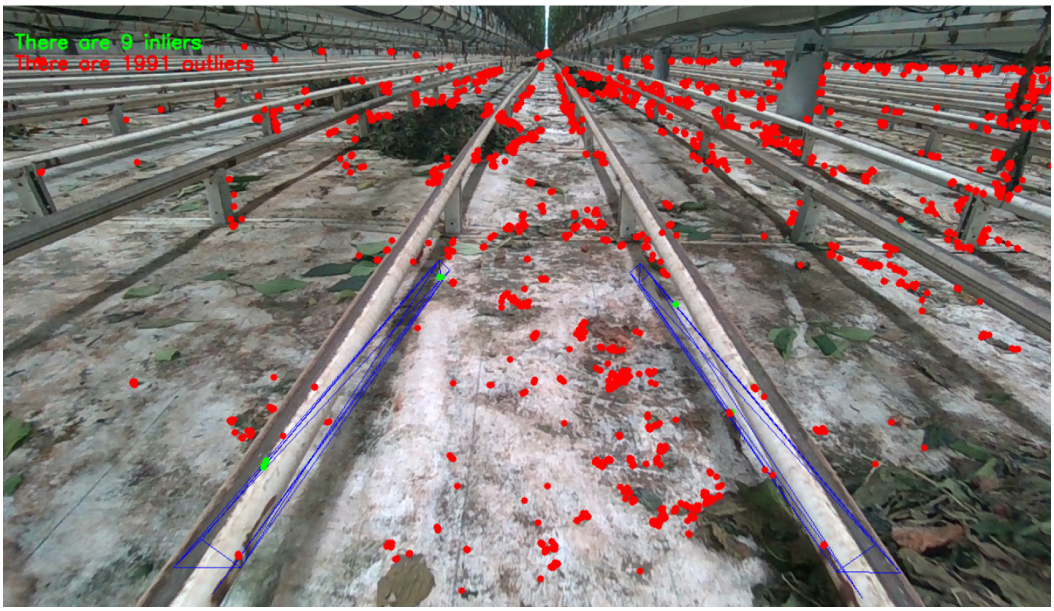
-

- Following links work on bounding box and with 3D Model
  - https://towardsdatascience.com/geometric-deep-learning-for-pose-estimation-6af45da05922 , https://github.com/vaishak2future/posefromkeypoints
  - https://docs.opencv.org/3.4/dc/d2c/tutorial_real_time_pose.html
- The last two links (Pose from key points) and Real time pose needs following:
  - Given a Bounding box, we can create a keypoint descriptors (Cropped Image with keypoint heatmaps overlayed/ orb features)
  - We need a 3D Model of the object we are interested in. Example: https://github.com/opencv/opencv/blob/4.x/samples/cpp/tutorial_code/calib3d/real_time_pose_estimation/src/main_registration.cpp . For this, we need a .ply file of (3D Model) of the object.
  - Another way to get a model: https://arxiv.org/pdf/1703.04670.pdf (Kinect Fusion - Given RGB-D image where depth is used only for ground truth. The ground truth object pose for each image was calculated by ICP with careful manual initialization. A complete 3D model of the gas canister was reconstructed using KinectFusion. Then, 10 keypoints were manually defined on the 3D model and projected to the images yielding ground truth keypoint locations in 2D.
- Current situation and what we are missing:
  - Given:
    - Bounding Box of the object. (i.e. 2D Image points) camera_T_pipe.
  - Task:

    - Localize the pipe i.e. find the position of the pipe in the real world. world_T_pipe.
- To convert pixel values of the pipe in the image to a real world pose, we need to know the scale between pixel to mm i.e. camera_T_world transformation and apply PnP to find the Transformation.
    - Establish point correspondence between 2d image pixels w.r.t. 3d world points.
        - Create a 3D Model of the pipe (.ply file) , then do feature extraction and flann matching with 3D Model to get point correspondence between 2d-3d points and do PnP to get the desired result. (Explore using Kinect Fusion, but Need scan of the pipe to create this) - https://github.com/opencv/opencv/tree/4.x/samples/cpp/tutorial_code/calib3d/real_time_pose_estimation
        - Semantic keypoints: Given RGB-D image where depth is used only for ground truth. The ground truth object pose for each image was calculated by ICP with careful manual initialization. A complete 3D model of the object was reconstructed using KinectFusion. Then, 10 keypoints were manually defined on the 3D model and projected to the images yielding ground truth keypoint locations in 2D. This is a learning based algorithm. https://towardsdatascience.com/geometric-deep-learning-for-pose-estimation-6af45da05922
        - Point correspondence between 2D Image and 3D Point cloud using keypoint descriptors for both the dataset. For training the model after keypoint extraction, we need to establish point correspondence which can be done if we have the camera's extrinsic and intrinsic parameters. No open source code

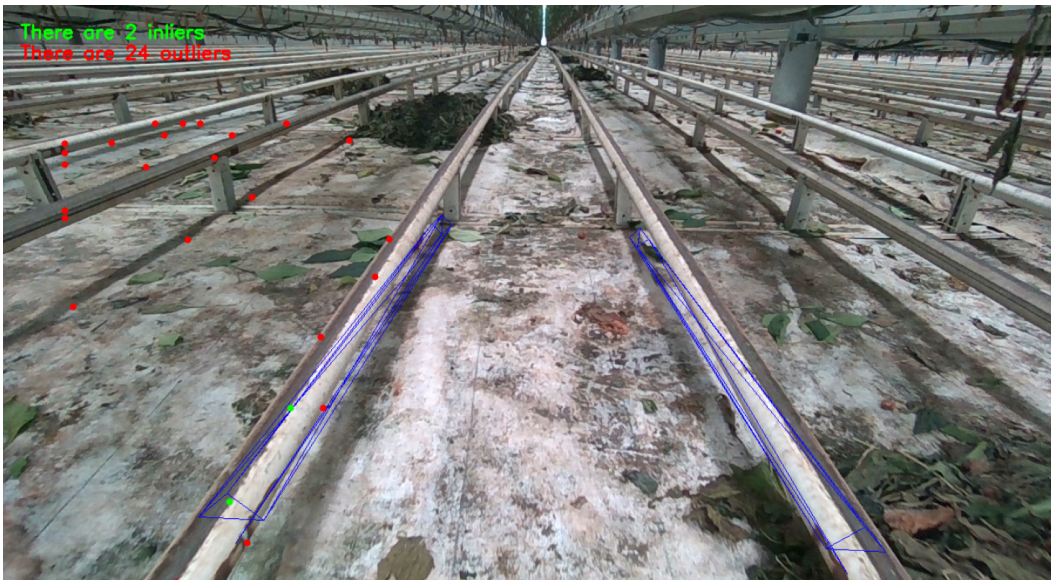
        - This can be done manually (by physically measuring the world w.r.t. pixel) but it will be tedious and might be different for each camera. Can be error prone.
    - Another way to look at the problem is as a slam problem. Instead of finding world_T_pipe, we find map_T_pipe. Map is generated using RTABMAP and we localize the robot in the map i.e. get map_T_base_link. From bounding box, for every pixel in it we have, camera_T_pipe. We also know base_link_T_camera. So we can now get map_T_pipe = map_T_camera * camera_T_pipe.
- Blue colored algorithm implementation:
    - Building and compiling code on Ubuntu 22.04 and run on custom 3D Model to create a mesh.
    - Step 1 - Creating a 3D Textured Model
        - Input:
            - Image of the pipe
            - 3D Model of the pipe (.ply)
            - Intrinsic Parameters: `color: [ 1280x720 p[648.812 360.262] f[643.008 642.201] Inverse Brown Conrady [-0.0562901 0.0673987 -0.00054218 0.000138015 -0.0220508] ]`
            - `depth: [ 848x480 p[424.16 239.665] f[429.381 429.381] Brown Conrady [0 0 0 0 0] ]`
        - Output: yml file (done)
    - Step 2 - Pose Estimation using loaded information (model detection)
    - Problem: The initial pipes.ply file sent had 720 vertices that wasn't possible to match and align the axis.
    - Created a new .ply file for two pipes with number of vertices to be 8 for each cylinder that means the top of the cylinder is assumed to be a square. By default, Blender (software), used 32 vertices for defining a circle but we don't need that.

- Using .ply, the model registration can get correspondences but the pipes are not thick enough to have a ton of orb features. This is a blocker. There are only 9 features inside the pipe and 1991 outliers.



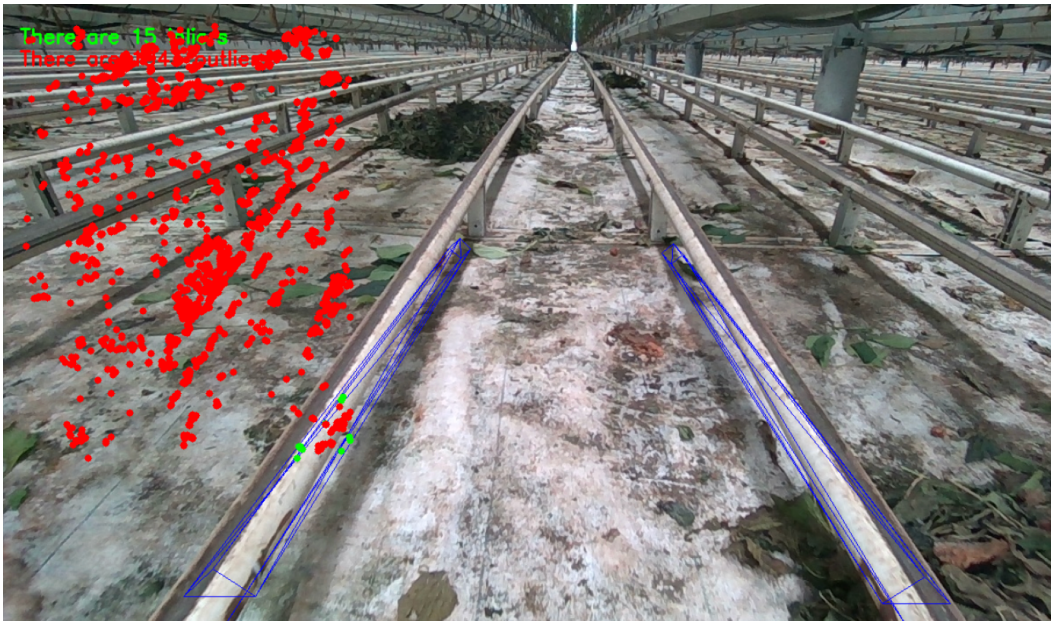
- Come up with a new approach to improvise
  - I decided to use only lines as features since pipe has lines. I implemented a line detector with a descriptor to use as features. The detector was detecting too many straight lines.
  - I added the code to use bounding box around the pipe (ideally this will come from yolo) and only detect line features inside the bounding box. This wasn't helpful as well since box is still big.
  - I added code to filter the detected lines based on the length of the lines. This helps in reducing the outliers but doesn't increase the inliners. Below ss with filtered lines

- 



- The output only has 2 inliners which isn’t enough for tracking.

- For bounding box, using ORB features also doesn't work.



- This algorithm doesn't seem to work for the use case of detecting pipe.

- Research new techniques, probably how to use depth information for pose estimation. (Similar to http://wiki.ros.org/find_object_2d)
  - We have depth/points i.e. sensor_msgs/PointCloud topic.
  - This algorithm needs an rgb image as well as a depth aligned image. This depth aligned image stores depth value at each pixel so the size should match and both the topics need to be in sync.
  - The algorithm uses Feature matching to detect object. Then uses the center of the detected object (bounding box) as well 3 more points inside the box (pixel points). For every pixel point, we can get XYZ coordinate using camera's info, pixel value as well as depth (mat) image. Then we apply PnP to get the tf/pose between camera_link and object
  - **Week of 27th feb:**

    To implement the same, the first step is to create a depth aligned image from point cloud points topic in the rosbag. The script is named convert_pointcloud_to_depth.cpp in find-pose package. It takes point cloud and camera's intrinsic parameter, convert it to pixels and add depth to it.

- Another way to convert to depth needed point cloud data which is organized(An *organized point cloud* resembles a 2-D matrix, with its data divided into rows and columns). The pointcloud available in the bag is unorganized(*unorganized point clouds* consist of a single stream of 3-D coordinates, each coordinate representing a single point). Conversion of unorganized pointcloud to organized lead to discrepancy in the data and was not in sync with rgb image data. This would have added another layer of potential errors. I wrote the code for this as well.
- The problem with this is that the depth images produced are not in sync with the rgb images. But we can start with whatever depth image we have.

- To estimate pose, a node takes two images, one rgb image with bounding box and a depth image corresponding to it. Take the measurement of the diagonal pixels from the bounding box along two axes and get the same pixels for depth value, apply pnp and estimate pose.

  In the following screenshot, the depth values are wrong so the pose is wrong.

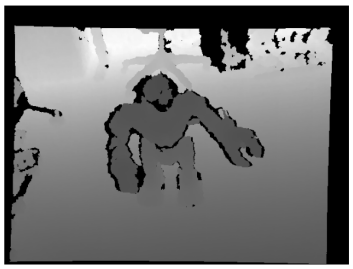
```
parallels@ubuntu-linux-22-04-desktop:/media/psf/freelancing/greenhouse/ros2_ws(main)$ ros2 run find-pose find_pose_node
[INFO] [1678087699.561783431] [find_object_2d]: object_prefix = object
[INFO] [1678087699.561968640] [find_object_2d]: pnp = true
depth pt: -5.76766e-38 y: 4.47781e-38 z: 9.40395e-38
depth pt: -3.15133e-39 y: 4.47781e-38 z: 9.40395e-38
depth pt: -5.08167e-38 y: -5.87346e-37 z: 1.51643e-36
depth pt: -2.30706e-37 y: -1.45694e-37 z: 3.76158e-37
depth pt: -7.60349e-39 y: 9.71048e-40 z: 2.35099e-38
depth pt: -2.94581e-41 y: -3.89421e-41 z: 1.84391e-40
depth img cols: 1280 rows: 720
Pose tvec x: -2.41649e-46 y: -4.03306e-46 z: -1.71245e-45
depth pt: -1.67481e-38 y: -3.04678e-39 z: 2.36943e-38
Pose x: 3.34357e-39 y: 5.58032e-39 z: 2.36943e-38
```

- Requirements:
    - /rgb/image_rect_color (sensor_msgs/Image)
    - /depth_registered/image_raw (sensor_msgs/Image)
    - /depth_registered/camera_info (sensor_msgs/CameraInfo)

2.4 depth_image_proc/point_cloud_xyzrgb

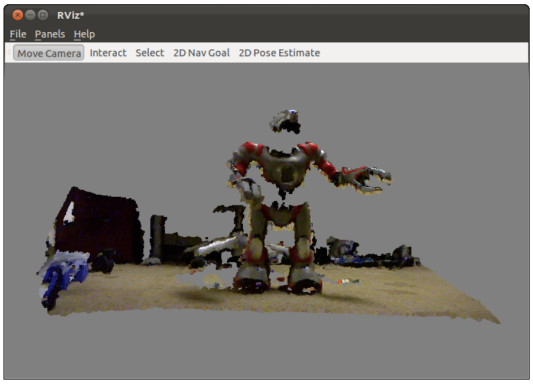
Nodelet to combine registered depth image and RGB image into XYZRGB point cloud.

Input **Output**



Registered depth image

RGB image



Color point cloud

    - I have colored point cloud and rgb image as input and I want a registered depth image.
- Researching available data from librealsense:
    - https://github.com/IntelRealSense/librealsense/tree/master/examples/measure

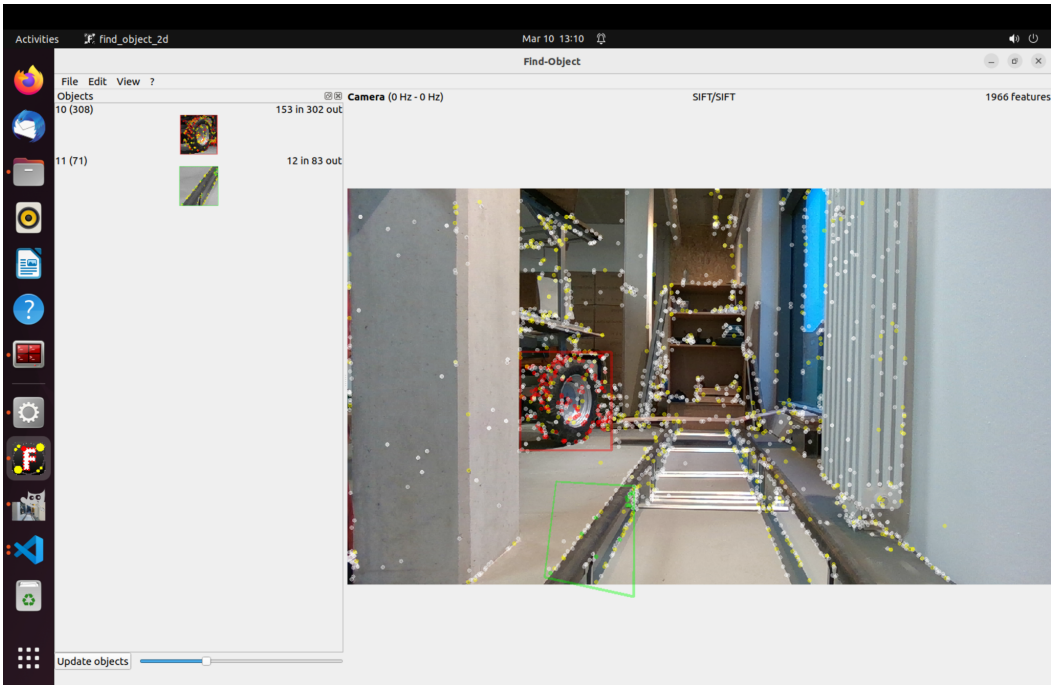
- Launch file:

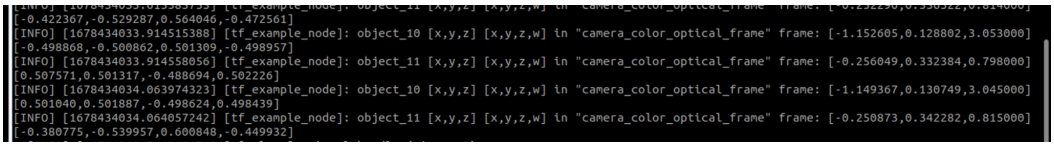
  ```
  roslaunch realsense2_camera rs_camera.launch align_depth:=true depth_width:=1280 depth_height:=720 enable_sync:=true enable_color:=true
  ```

  ```
  $ rostopic hz /camera/color/image_raw
  ```
- ```
  $ rostopic hz /camera/aligned_depth_to_color/image_raw
  ```
- ```
  $ rostopic hz /camera/color/camera_info
  ```

Week of 6th March:

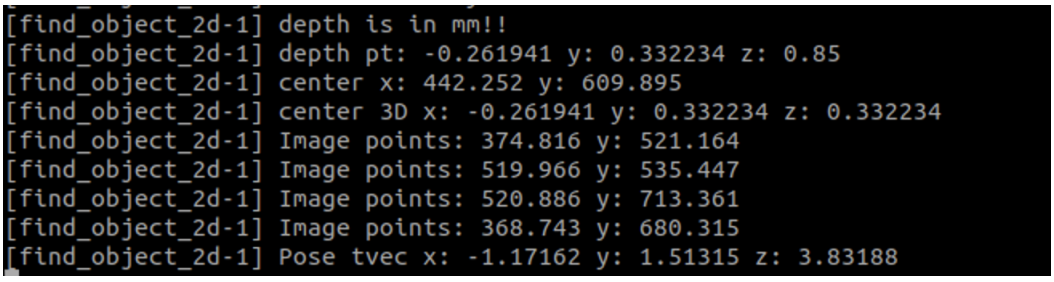
- Got a new bag from Zed2i (second bag) 2023-03-07.bag
- Converted bag data to rgb images and depth images. Verified depth images are in sync and data satisfies the code.
- Researched on Zed2i's object detection module. (It also does yolov5/yolov4 model training) and pose estimation. The API is not open source.
- The bag provided had a different image dimensions i.e. 640x360 than the previous bag.
- - This data seems to be from a custom setup similar to the one at greenhouse. I am assuming the distance between two pipes is the same as in the original environment and the dimensions of the pipe are same.
- - The data was trained on 1280x720 images as well as in a different environment. Since both the dimensions as well as the environment are a lot different, I would need to retrain the yolo module to get a different set of weights. For now, I am assuming a bounding box for the pipe manually. If the poc is successful, then we can think of retraining.
- Got a new bag from d455 (third bag) 2023-03-09-18-08-11.bag
- Converted bag to rgb and depth images.
- Demo of pose estimation using features. (Not very reliable as features are less.)

- 

- 

- Current pose estimation code takes in rgb image, depth image, camera info, bounding boxes of the object. Gives out position of the object.

  Screenshot from features based code:



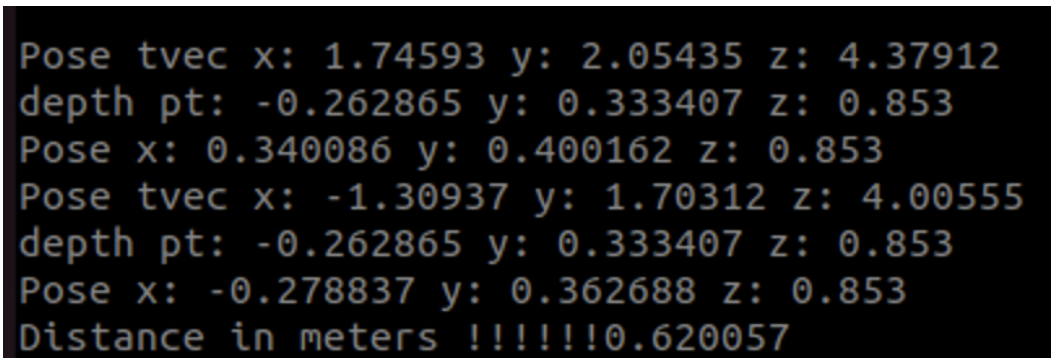
Screenshot from assumed yolo bounding box code:

```
parallels@ubuntu-linux-22-04-desktop:/media/psf/freelancing/greenhouse/ros2_ws(main)$ ros2 run find-pose find_pose_node
2023-03-10 16:14:39.854 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7412: open_and_lock_file failed -> Function open_port_internal
[INFO] [1678445079.864881151] [find_object_2d]: object_prefix = object
[INFO] [1678445079.864994692] [find_object_2d]: pnp = true
depthconstant: 0.00156033
Pose tvec x: 1.74593 y: 2.05435 z: 4.37912
cx: 639.5 cy: 359.5
fx: 640.889 fy: 640.889
depth is in mm!!
depth pt: -0.0785823 y: 0.0996702 z: 0.255
center x: 442.252 y: 609.895
center 3D x: -0.0785823 y: 0.0996702 z: 0.0996702
Pose x: 0.101667 y: 0.119626 z: 0.255
Done!!!!!!!!!!!
depthconstant: 0.00156033
Pose tvec x: -1.30937 y: 1.70312 z: 4.00555
cx: 639.5 cy: 359.5
fx: 640.889 fy: 640.889
depth is in mm!!
depth pt: -0.0785823 y: 0.0996702 z: 0.255
center x: 442.252 y: 609.895
center 3D x: -0.0785823 y: 0.0996702 z: 0.0996702
Pose x: -0.0833569 y: 0.108424 z: 0.255
Done!!!!!!!!!!!
```

- From above two screenshots, the camera intrinsic, bounding box as well as rgb images have matching data. Currently, I was reading the depth image I converted the bag into. If I do that then, depth is different from reading directly from a bag.
- Ran yolov3 on new dataset, but the detection doesn't work since the scene is very different from the previous dataset. Command to run detection: python detect.py --img 1280 --source ../pipe-dataset/validate/images/stereo_image103.jpeg --weights runs/train/exp14/weights/best.pt
- 
- Should i re-train the model or not?
- Run yolo as a ros node on a bag. https://github.com/ros2/openrobotics_darknet_ros
- Need to hook up yolo's bounding box as well as subscribing to rgb and depth topic, tf and camera info.

Week of March 13th, 2023:

- Update find-pose code to read data directly from the bag. And assuming bounding box to be consistent as yolo isn't working in the new data. As seen in following screenshot, the distance in meters between two pipes is about 0.62meters. The physical distance between pipe is 0.551m so currently there is an error of 0.059meters i.e. approx 6cm.



- Next is to hook up yolo on a bag.
- Build darknet_vendor and darknet_ros2. Both are ros nodes.
    - darknet_ros2: https://github.com/ros2/openrobotics_darknet_ros
    - Darknet_vendor: https://github.com/ros2/darknet_vendor
    - Darknet_ros (ros2 branch): https://github.com/leggedrobotics/darknet_ros
    - Darknet_ros_fp16: https://github.com/Ar-Ray-code/darknet_ros_fp16 (this one works)
- Build steps:
    - First build darknet_vendor by using the following command:
    - Cd darknet_vendor
    - Mkdir build && cd build
    - cmake -DDARKNET_CUDA=Off -DDARKNET_OPENCV=Off ..
    - colcon build --cmake-args -DCMAKE_BUILD_TYPE=Release -DDARKNET_CUDA=Off -DDARKNET_OPENCV=Off --packages-select darknet_vendor
    - Install vision_msgs by running: sudo apt-get install ros-humble-vision-msgs*

    - Now build darknet_ros2: colcon build --packages-select openrobotics_darknet_ros
- There were few build errors because vision_msgs were updated and this code was old so updated the code to use latest ros2 msgs and function calls.
- Darknet ros2:
    - Subscribers: ~/images (type sensor_msgs/msg/Image) - Input mages to feed to the detector
    - Publishers: ~/detections (type vision_msgs/msg/Detection2DArray) - Objects detected in an image (if any)
    - Parameters
        - network.config - a path to a file describing a darknet detector network
        - network.weights - a path to a file with weights for the given network
        - network.class_names - a path to a file with names of classes the network can detect (1 per line)
        - detection.threshold - Minimum probability of a detection to be published
        - detection.nms_threshold - Non-maximal Suppression threshold - controls filtering of overlapping boxes
- Wrote a launch file and updated the code to take in parameters properly via yaml.
- Problems in Darknet ros2:
    - It uses a .cfg file for the yolo3 structure which is a very old format. Newer version uses a .yaml file and currently one to one conversion isn't supported.
    - Even after getting a .cfg file similar to yaml file, and writing a launch file with parameters, I run into the issue that 'Argument list too long' and haven't been able to resolve it yet.
- Darknet ROS:
    - This repository also has a branch for ros2. But it is based on and opencv version before 4. And all the code for opencv is outdated. Especially the use of IplImage as a structure to hold images

- Darknet_ros_fp16: https://github.com/Ar-Ray-code/darknet_ros_fp16 (this one works)
    - Updated CMakeLists.txt to build the code properly and in correct configuration based on the system. (cuda off for now and yolov3 is built)
    - Updated ros.yaml and darknet_ros/launch/darknet_ros.launch.py to update topics
    - Following new files for cfg and weights:
        - darknet_ros/config/pipe_yolo3.yaml
        - darknet_ros/launch/pipe_detection.launch.py
        - darknet_ros/yolo_network_config/cfg/pipe_yolo3.cfg
        - darknet_ros/yolo_network_config/cfg/yolov3_pipe.cfg
        - darknet_ros/yolo_network_config/weights/pipe_yolo3.pt
    - Darknet builds and works on system on sample dataset.
    - Ran darknet on pipe dataset with best weights from yolov3 and config.

      Run command: ros2 launch darknet_ros pipe_detection.launch.py

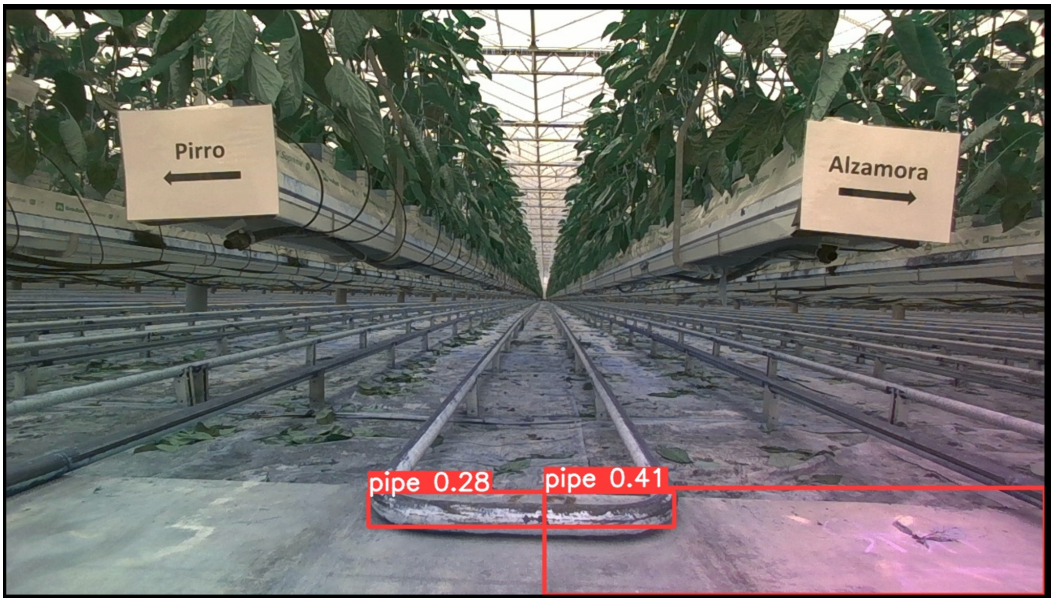
      Build command: colcon build --packages-select darknet_ros

      There are few issues with it:
        - Fps is very low. A FPS of 0.2 means that only 0.2 (or 1/5) of a frame is displayed per second, which is an extremely low frame rate. This might be due to not running on gpu and images being very large. It might be that cpu is being hogged by threads so we can try different settings to increase it.
        - The yolov3 outputs weight file in .pt format and config of network in yaml format. Whereas Darknet has weights in .weight format and config in .cfg format. Even after converting the files to desired format, the detection topic is empty.

Week of March 20th, 2023:

- Wrote code called detect_on_bag.py in yolov3. This node subscribes to an image topic, run detection on it and publishes an image with a bounding box on it. (Monday)
- Received bag 2023-03-20-11-17-20.bag

- Updated the bounding box structure and now publishing is correct. Wrote code in find-pose rosnode to listen to the bounding box at the same timestamp as image, depth image and camera info and estimate pose.
- The detection on bag is still quite slow. The rate on bag is around 2.7second for 1 image i.e. 0.4fps. GPU was giving about 20fps so I think this should not be a blocker.
- When I ran the entire pipeline on the latest bag, the yolo detection behaved poorly as follows:

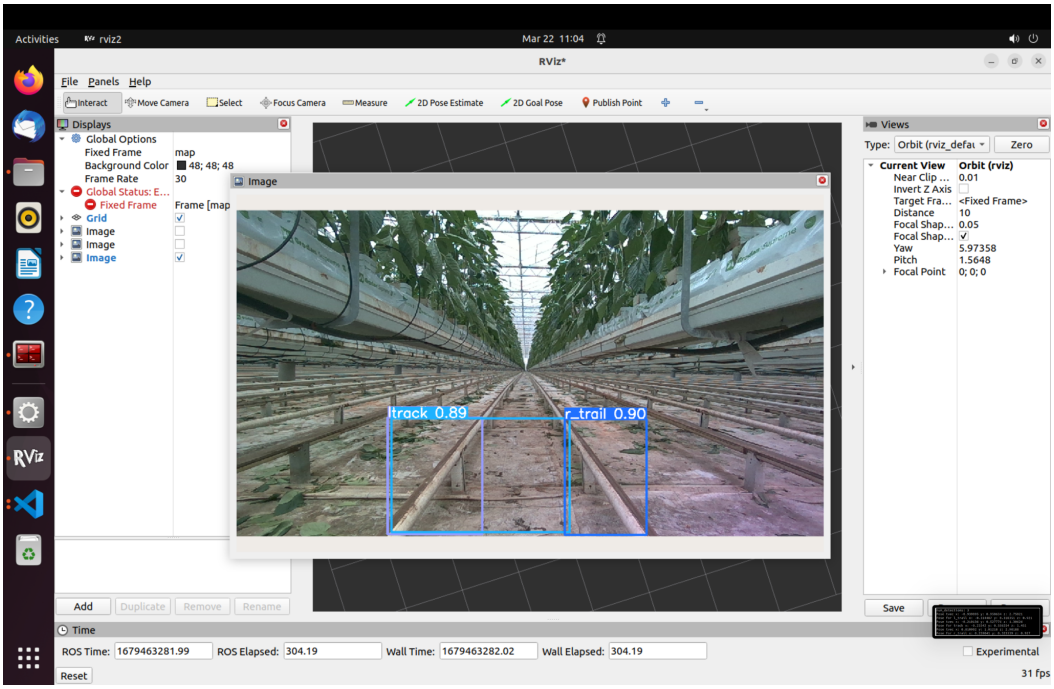


- Since the camera is at a different height as compared to data collected previously, and there weren't any pictures from this view, the detections are quite wrong.

- Retrained the model by combining new data for training. train image: 303/313, validate: 53
- Fixed the time sync between data to process and ran the entire pipeline for pose estimation.

  When we see a camera moving in open space, the z axis is changing from 3.2 meters to 2.8 meters which means the camera was moved about 0.4 meters.
- When we see the left and right pipes, and calculate the distance between two poses, we get that distance is about 0.555meters (555mm). The physical distance between two pipes were 550mm.

```
num_detections: 3
Pose for track x: -0.215547 y: 0.622883 z: 1.424
Pose for l_trail x: -0.317584 y: 0.343902 z: 0.907
Pose for r_trail x: 0.236329 y: 0.354132 z: 0.927
Distance in meters !!!!!!0.554368
```

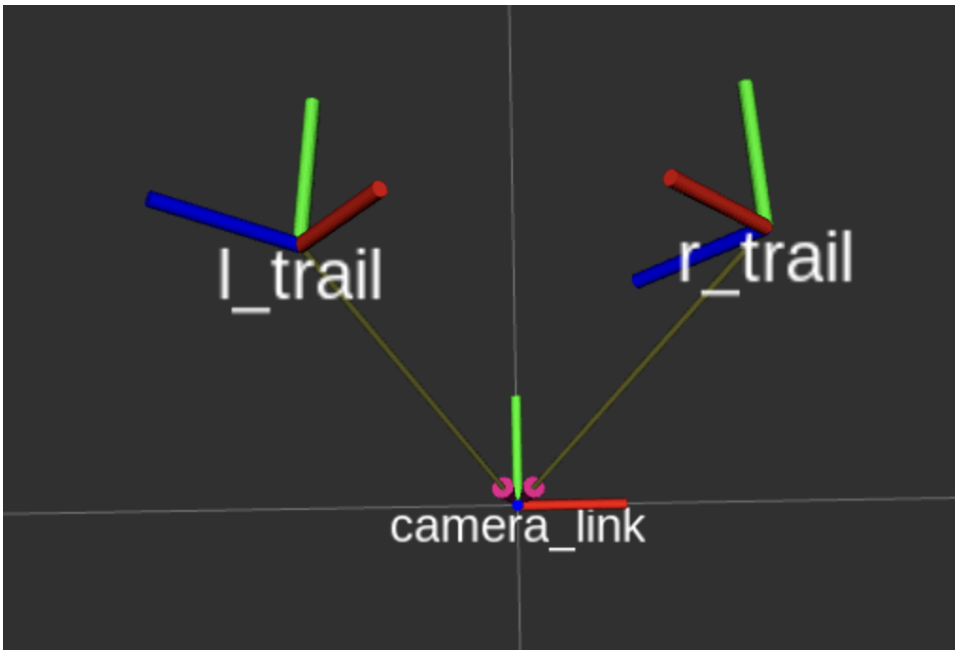


- Commands:
  - $ ros2 run find-pose find_pose_node
  - /media/psf/freelancing/greenhouse/ros2_ws(main)$ python3 src/yolov3/detect_on_bag.py

- Weights from src/yolov3/runs/train/exp15/weights/best.pt

Week of March 27th, 2023:

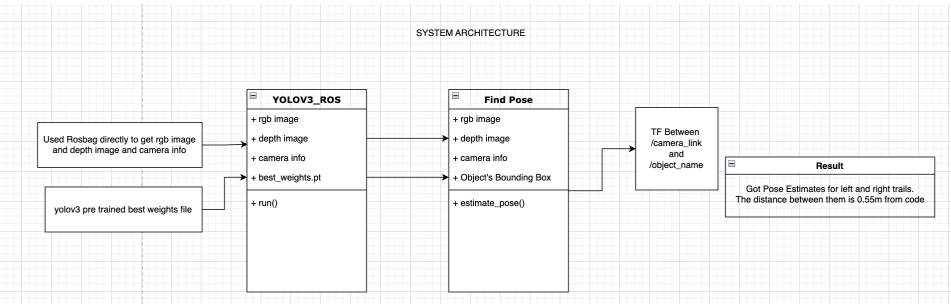
- Updated code to add tf broadcaster.
- TF's x red, y green and z blue.
  - Camera link: top left corner of the image/object(rectangle) in image is 0,0. Z goes out of the image towards 3d object, x goes to left to right and y goes from up to down.
  - Object link: x axis going front of the object, with z up and y left.



-
```
callback!!!!!!!
num_detections: 3
Pose for track x: -0.189824 y: 0.590806 z: 1.403
Pose for l_trail x: -0.298458 y: 0.329342 z: 0.886
Pose for r_trail x: 0.26418 y: 0.342167 z: 0.939
broadcast tf
Euclidean distance between two pipes in meters !!!!!!0.565274
Angular distance between two pipes in meters !!!!!!3.03554
```

  - TODO: Accuracy estimation methods:
    - Ground truth: Either manually measure the distance between camera and pipes or use motion capture and compare with the pose from the algorithm.

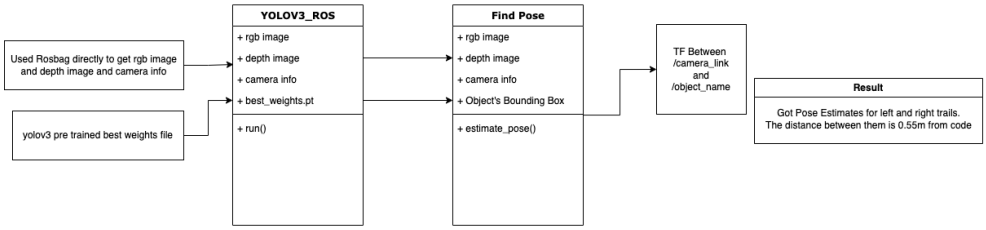
        - Plot mean error, median error and standard deviation over time or distance.
    - Using point cloud data to make sure tf gets updated as we move.
- Handover tasks.
- Final System Architecture:



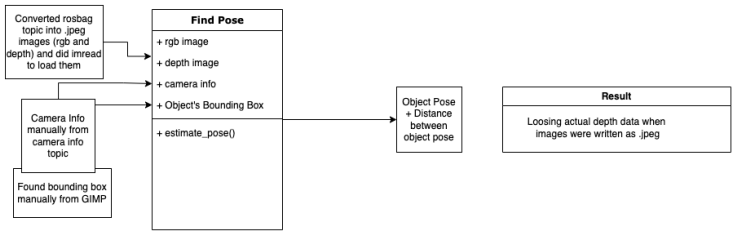
- Final Node names:
    - Find-pose: Module that does pose estimation based on detected bounding box. Output is the tf between the camera_link and object_detected.
    - Yolov3_ros: Module that does object detection and sync incoming data. The output is detected object's bounding box.

Block Diagrams:

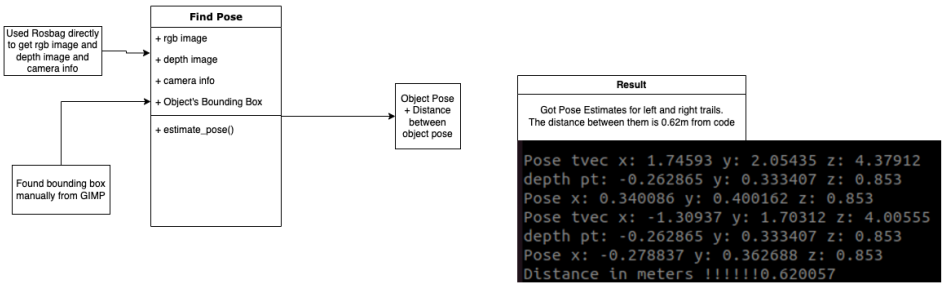
SYSTEM ARCHITECTURE



POC 1



POC 2



TODO:

- Run yolov3 on the bag with new weights.
- ~~Given 3D Cad model, use opencv pose estimation (WIP)~~
- Need to decide evaluation of the algorithms
    - Accuracy
    - Pose Error Translation as well as rotational
-

Approach

- Using a NEURAL NETWORK based object detection technique like YOLOv7 to get a bounding box with center and corners.
- Using a classical approach to detect pipes in the environment (computer vision/ pcl based) and get a pose.

Validation:

- diameter of the pipe: 51mm, distance between the two pipes: 550mm

Deliverables:

- ReadME with instructions on how to build and run the software.
- Clean code
-

Next Goal

- Need a visual slam based mapping algorithm to create a map.
    - Once we have a map, we need to convert the position from one of the two approaches to map co-ordinates for localization.