Laavanye Bahl

Team D: CuBi

Teammates:
Jorge Anton, Paulo Camasmie, Changshen Shen, Nithin Subbiah Meganathan

ILR03

March 7, 2019

Individual Progress

The main focus for this review was to accelerate the task of object detection by taking the previous work done on static point clouds and single frames and developing a whole end-to-end pipeline for real-time operation. I worked mainly on the following:

- Develop a ROS node to get continuous stream of point clouds. Faced the problem of slow speed in python and shifted to C++.
- Identified and used the correct message type for point cloud data from several options.
- Reduced the space complexity and increased the speed three times as compared to the previous module in python by efficiently using pointers and references wherever possible as a single point cloud message is very big and dense.
- A lot of trial and testing to set the correct hyper parameters for the various functions in the pipeline. E.g. leaf size for voxel filter down-sampling, number of neighbors for mean in statistical outlier removal, distance threshold in RANSAC, etc.
- Developed modular code and structure in ROS (include files, launch files, helper functions, boost and eigen optimizations) to make a good base for vision tasks and the project in general.
- Started work on making a urdf for the robot so that we can find transforms of the point cloud to various frames.
- Started work on fitting bounding boxes (Axis aligned bounding boxes or AABB).

The demo can be seen in this video and in **Fig. 1a.** and **Fig. 1b.** below.

Fig. 1a. First person view of the camera setup

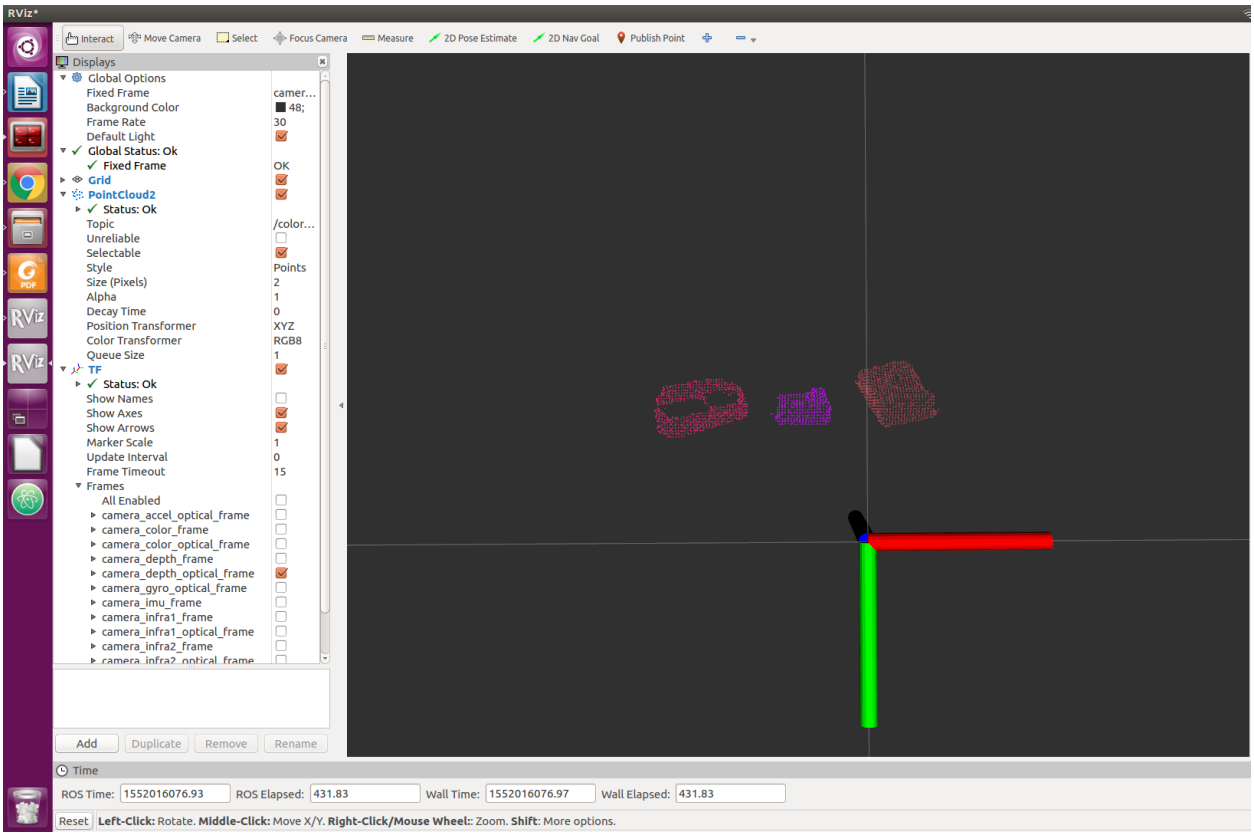


Fig. 1b. 3D clustered output in real time

Challenges

Individual challenges:

The first major challenge I faced was of the extremely slow point cloud input stream while using external python packages/ libraries like python-pcl. I had my initial code for object detection on static single frame point cloud written in python. So, I rewrote the whole code in C++.

Then the problem I faced in C++ was to identify the best message type for receiving and for processing the point cloud.
There were several options:

- sensor_msgs::PointCloud
- sensor_msgs::PointCloud2
- pcl::PointCloud<T>
- pcl::PCLPointCloud2

Since the point cloud was a colored one (XYZRGB) and not just the simple one (XYZ), I faced a huge challenge of converting the point cloud received from sensor_msgs::PointCloud2 to the native pcl::PointCloud<pcl::PointXYZRGB> or pcl::PCLPointCloud2.
This seems to be specific to intel realsense and the solutions on the internet did not work.
After a lot of trials, I figured out that pcl::PCLPointCloud2 could be used to subscribe directly to the stream and could be used for processing as well instead of the native pcl::PointCloud<pcl::PointXYZRGB> format. But, again the conversion from pcl::PCLPointCloud2 to pcl::PointCloud<pcl::PointXYZRGB> resulted in loss of color. So, I used pcl::PCLPointCloud2 for most of the initial processing then finally used pcl::PointCloud<pcl::PointXYZRGB> for functions like euclidean clustering, where color information did not matter and it was the only supported format.

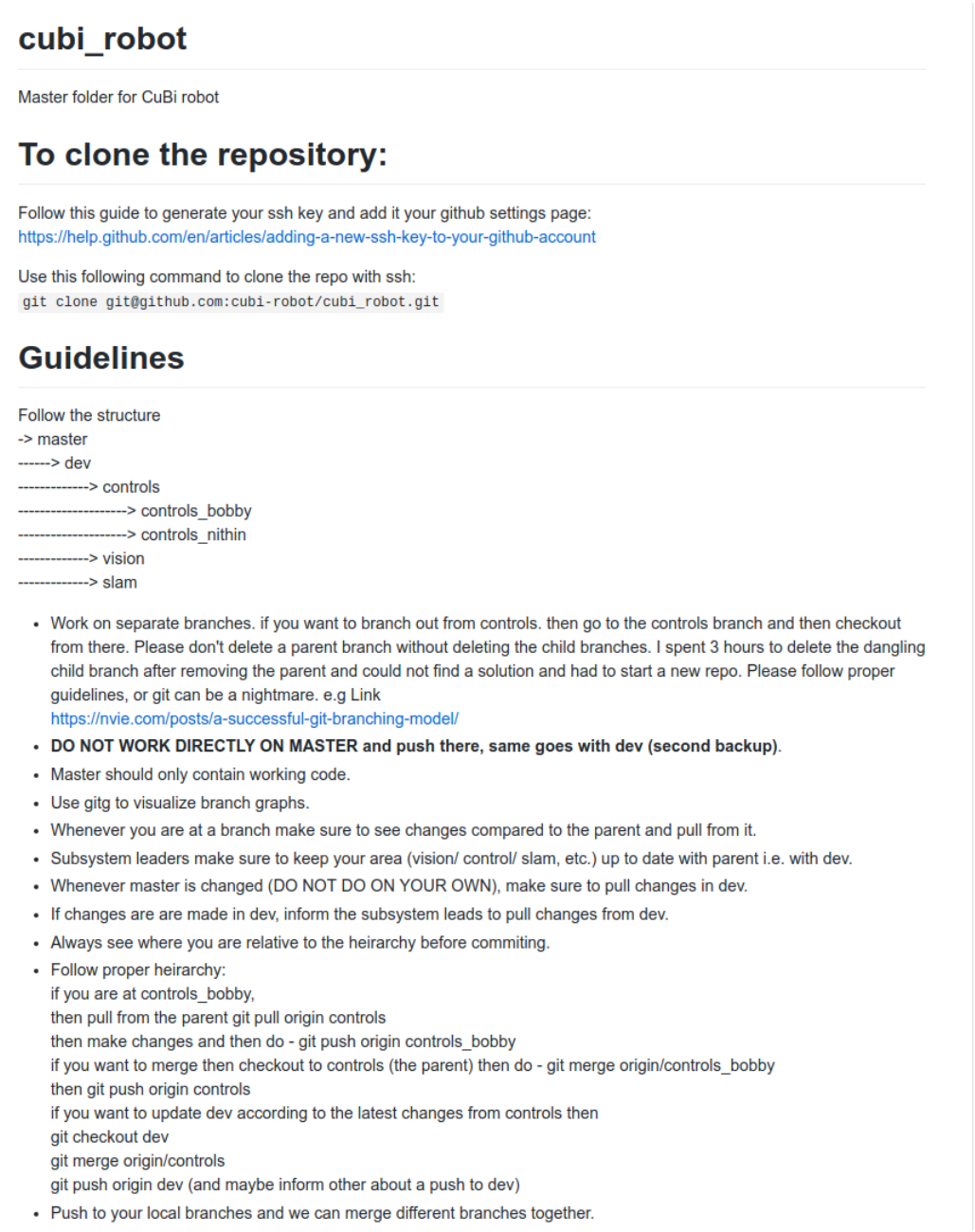
Team challenges:

The major challenge for the team this time was making the dynamixel motors work. We tried a lot of solutions, read multiple blogs and articles, but could not make it work. After talking to some experts, we got to know that every six months there is a new update for the motors. Hence, we have to be careful with them. We are talking to multiple people for help, working ourselves for solutions and also considering buying new ones.

The next big challenge we were facing as a team was how to collaborate in an organized way and be up to date with the increasing code base. I have described how I tackled this problem in the starting of the Teamwork section.

Teamwork

Most important work I did for teamwork was to take the initiative to address the problem of collaboration and set up a well organized git repository with proper guidelines and branch structure as can be seen in the screen-shot in **Fig. 2.**



cubi_robot

Master folder for CuBi robot

To clone the repository:

Follow this guide to generate your ssh key and add it your github settings page:
https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account

Use this following command to clone the repo with ssh:
`git clone git@github.com:cubi-robot/cubi_robot.git`

Guidelines

Follow the structure
-> master
------> dev
----------> controls
----------> controls_bobby
----------> controls_nithin
----------> vision
----------> slam

- Work on separate branches. if you want to branch out from controls. then go to the controls branch and then checkout from there. Please don't delete a parent branch without deleting the child branches. I spent 3 hours to delete the dangling child branch after removing the parent and could not find a solution and had to start a new repo. Please follow proper guidelines, or git can be a nightmare. e.g Link
  https://nvie.com/posts/a-successful-git-branching-model/
- **DO NOT WORK DIRECTLY ON MASTER and push there, same goes with dev (second backup)**.
- Master should only contain working code.
- Use gitg to visualize branch graphs.
- Whenever you are at a branch make sure to see changes compared to the parent and pull from it.
- Subsystem leaders make sure to keep your area (vision/ control/ slam, etc.) up to date with parent i.e. with dev.
- Whenever master is changed (DO NOT DO ON YOUR OWN), make sure to pull changes in dev.
- If changes are are made in dev, inform the subsystem leads to pull changes from dev.
- Always see where you are relative to the heirarchy before commiting.
- Follow proper heirarchy:
  if you are at controls_bobby,
  then pull from the parent git pull origin controls
  then make changes and then do - git push origin controls_bobby
  if you want to merge then checkout to controls (the parent) then do - git merge origin/controls_bobby
  then git push origin controls
  if you want to update dev according to the latest changes from controls then
  git checkout dev
  git merge origin/controls
  git push origin dev (and maybe inform other about a push to dev)
- Push to your local branches and we can merge different branches together.

Fig 2. Initial guidelines for GitHub repository

I met and explained this to the whole team, so that everyone is on the same page and can start using good practices.

In our team meeting, I explained the whole team about the vision pipeline and what each function does, where I was stuck and get some feedback.

Following describes the work done by the team members and how I interacted with them:

Paulo:
Finalized the design of the manipulator and its assembly as shown in **Fig. 3**. He extended the tray by 20mm to accommodate two toys at a time as shown in **Fig. 4**. He also did calculations to measure the maximum torque required and the estimated payload for the tray.
I interacted with him about the effects of the potential arm designs on the positioning of the camera and LiDAR. I showed him a couple of options for the mounts of the camera and will be collaborating with him for the same before next review.

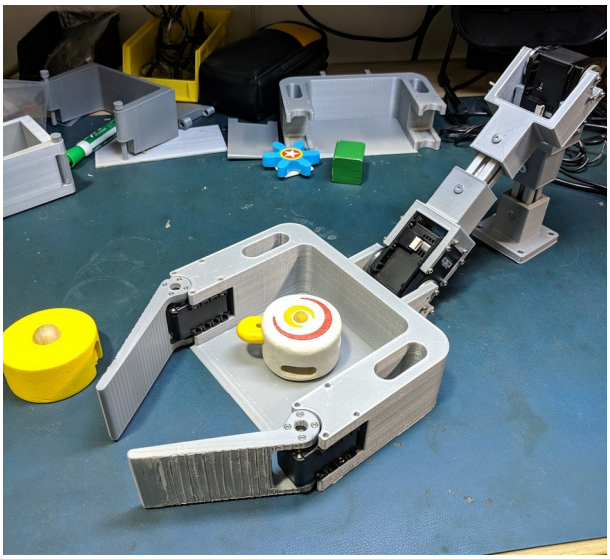


Fig. 3. Final robot assembly (Credits: Paulo)

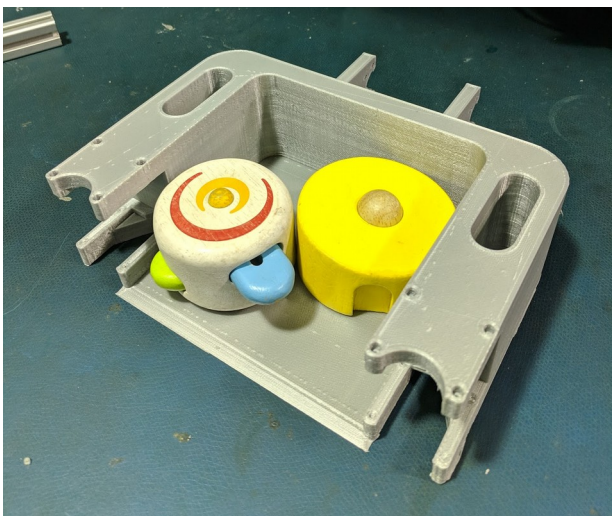


Fig. 4. 20mm bigger tray (Credits: Paulo)

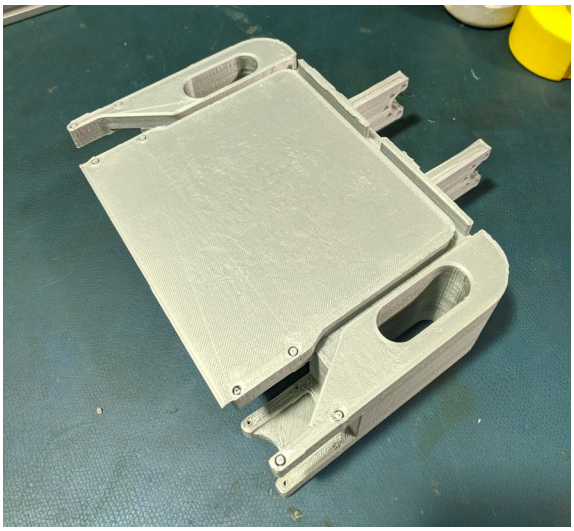


Fig. 5. Provisions for wiring (Credits: Paulo)

Bobby:
Made wires and connectors for the electronic system and wrote ROS code for controlling dynamixels using joystick. He also did most of the work for PCB. He also helped Nithin and Jorge on resolving motor issues.
I interacted with him about the issues of Jetson. He also provided valuable feedback on the vision pipeline.

Jorge and Nithin:
Worked together on controlling the motors and resolving the issues by multiple ways (ROS, Mixel, GUI).
They helped me by asking good questions about the vision pipeline and validating it.

Future Work

Individual plans:

- Fit an oriented bounding box (OBB) instead of AABB around the clustered objects.
- Do pose estimation of the objects.
- Validate the toys with the current vision pipeline.
- Make basic urdf of robot in order to calculate the transforms and view point cloud from deferent frames.
- Try another technique for RANSAC- find perpendicular plane to z axis rather than fitting for the biggest planar surface and using that ground removal.

If time permits:

- Maintain a list of current objects and try to assign same color to clusters rather than random colors at every frame.
- Research on using CUDA for processing on GPU.
- Research on optimizing and further improving the pipeline like using temporal information. E.g. average over n previous frames to make the 3D reconstruction more robust.

Team plans:

- Resolve the issues with motors and control them with ROS.
- Start work on SLAM.
- Mounting of sensors and manipulator.
- Validation of payload.