Control Robotics and Machine Learning Laboratory

REPLAB full technical report -

Control Robotics and Machine Learning Laboratory Technion.

Project Name: REPLAB

Instructor: Orr Krupnik

Students: Raveh Ben Simon & Omer Cohen

Table of contents

Images

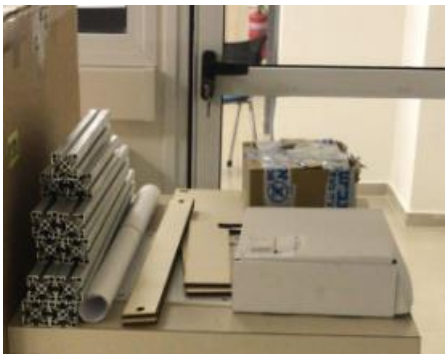

Figure 1

Figure 2

Figure 3

Figure 4

Figure 5

Abstract

The project's goal is to replicate a standard, reproducible, and cheap environment for benchmarking robotic arm object grasping algorithms. The original environment was created in Berkley and reviewed in the article: "REPLAB: A Reproducible Low-Cost Arm Benchmark Platform for Robotic Learning" by Brian Yang, Jesse Zhang, Vitchyr Pong, Sergey Levine , and Dinesh Jayaraman.

The standard REPLAB cell includes an arena with a wooden base, a cage comprised of metal beams, WIDOWX MK II robotic arm of TrossenRobotics, and a 3D camera – Intel RealSense D435. The camera is the only difference from the original environment.

In this report we will entail our efforts to replicate the experiment and results of the aforementioned article, while using the Intel Realsense D435 instead of the SR300.

The full details of the original environment are described in REPLAB's official website: https://sites.google.com/view/replab/home

Our Docker image that has the D435 camera integrated into it: https://hub.docker.com/repository/docker/ravehbs/replab_technion_realsense_d435

Our Github which contains all the required code changes: https://github.com/RavehBS/replab

Hardware setup

Building the robot - WidowX MK II

the instructions given by Trossen Robotics' website (https://learn.trossenrobotics.com/interbotix/robot-arms/widowx-arm.html) were quite thorough, we advise paying special attention to several details:

Hardware

a. First, it is important to lay out all the parts received in the Kit, count them, and make sure that all are accounted for according to the manifest in the assembly guide.
   For example, we found that our Turbofuse was dry and that the "Top Reverse Bracket" had a missing hole. We were advised by TrossenRobotics that any Loctite should suffice instead of the Turbofuse.

b. Make sure to peel all the stickers. The Replab website's images shows a constructed arm whose base is brownish wood. This is actually a sticker pasted on top of the base and should be removed. It will prevent proper assembly of the robot.

c. Servo configuration:
   TrossenRobotics recommends using the DynaManager to configure the servos before assembly. At the time of our assembly some of the servos were not able to interact with the software. This was resolved by asking TrossenRobotics' support (and receiving) the Dynamixel U2D2 adapter through which the configuration of the servos was possible.

d. Servo alignment:
   i. Make sure the horns are aligned to the servo on assembly (the dot on the horn aligned to the notch on the servo).

ii. When connecting parts to the horn, make sure that the servo horn is aligned (single dot facing the top of the servo)

iii. mark the front of the base (we marked with tape) and make sure all installations are done according to the required orientation in relation to the front.

Failing to make sure these are done properly will lead the WidowX arm to fail the final assembly test described in section "Assembly Test".

Software setup

Use getting started guide from here:

https://learn.trossenrobotics.com/arbotix/arbotix-quick-start.html

a. note that the FTDI drivers is likely working, and does not need to install:

   run the following command in the host's terminal:

   `"ls /dev | grep ttyU".`

   If the output received is: `"ttyUSB0",` then the host machine detected the connection properly and you may proceed.

b. Common errors we encountered:

   i. Arduino IDE:

      'Tools->Serial Port' is grayed out.

      Solution: The issue is that Arduino IDE does not have permissions to access the serial ports. to resolve, open the Arduino using sudo: `"sudo arduino".`

   ii. Failure to upload a sketch (for example the blink sketch) and getting the following error:

      `"libusb-0.1.so.4: cannot open shared object file: No such file or directory".`

      Solution: install a usb library with this command:

      `"sudo apt-get install libusb-0.1-4".`

Assembly Test

Before connecting the arm to the cage, run the test mentioned here:
https://learn.trossenrobotics.com/18-interbotix/robot-arms/widowx-robot-arm/27-widowx-robot-arm-build-check.html
Make sure that the arm is performing the exact same movements, with the same posture.

If the final test has passed correctly, the initial setup of the WidowX arm is done!

to proceed forward, Load the ROS sketch onto the Arbotix controller:
In the Arduino program use the following menus:
File->Sketchbook->arbotixSketches->ros.

Experiment environment

Setting up Docker program is done through official website:
https://docs.docker.com/
We recommend studying the concept of Docker and its basic functions such as: `run, exec, ps, images, commit, tag.`
There are plenty of blogs and material online regarding this subject.

after docker setup, get the docker image as mentioned in:
https://sites.google.com/view/replab/home
or use our docker image:
https://hub.docker.com/repository/docker/ravehbs/replab_technion_realsense_d435

In order to use all the functionality of the REPLAB project in docker, you must run the image using the following commands on the host machine:

Give permission to docker to access display (this must be done every time the host is rebooted):
command: `"xhost +local:docker"`
expected output: `"non-network local connections being added to access control list"`

Invoke the docker image:
The command provided on the official Replab website is not sufficient for using all of the programs inside the docker container. We recommend invoking the docker with the following command:

```
docker run \
--runtime=nvidia \
```

```
    -e NVIDIA_DRIVER_CAPABILITIES=graphics,compute,utility \
    -e NVIDIA_VISIBLE_DEVICES=all -it \
    -e DISPLAY=$DISPLAY \
    -v /$HOME/.X11-unix/:/root/.X11-unix:rw \
    --net=host \
    --env="DISPLAY" \
    --volume="$HOME/.Xauthority:/root/.Xauthority:rw" \
    --device=/dev/video0 --device=/dev/video1 --
device=/dev/video2 \
    --device=/dev/ttyUSB0 \
    --ipc=host \
    --privileged \
    <docker image name>
```

We strongly advise to understand and read about each of these flags for running the image, as they each play a role in specific functionalities required by REPLAB.
These flags can also help solve other issues that may pop up while using docker (for example sharing a volume between the docker and the host machine).

Customizing REPLAB docker:
At this point we have achieved a working docker image and a functioning WidowX arm.
If you are using the original docker given by the REPLAB team, follow these steps to customize the environment for use. We recommend using our docker image which already contains all these setups.
From this point forward we will lay out our efforts to customize the docker image for use with the Intel RealSense D435 camera, instead of the SR-300 suggested by the original paper.

Install realsense 2.0 Drivers on the docker image:

Follow the instructions laid out in step 1 and 2 on method 2 on the following link: https://github.com/IntelRealSense/realsense-ros

After finishing installation, check that the camera feed is working properly using the command `"realsenseviewer"` from within the docker. If this is working properly, make sure to commit the docker image for later use.

Common errors:

Error: Timeouts, connection not found and miscellaneous "not found" errors

Solution: Many of these errors are a result of a faulty USB outlet in the computer. Make sure to use USB3.0 outlets and cables when connecting to the D435.

Error: `"select timeout"` error when running cam_align.py or commander_human.py.

Solution: run the following commands on the host machine:

```
"sudo rmmod uvcvideo"
"sudo modprobe uvcvideo nodrop=1 timeout=5000"
```

Code Changes required to tailor the environment for the D435 camera:

Start.sh script:

change start.sh script to use rs_rgbd.launch as a launch file for the camera.

Change topic names in:

/root/ros_ws/src/replab/replab_core/src/replab_core/commander_human.py line 202:

```
gui =
ClickWindow("/camera/color/image_raw","sr300_img_clickpoints",
bgr_to_rgb)
```

/replab_core/src/replab_core/click_window.py

line 198:

```
gui = ClickWindow("/camera/color/image_raw","sr300_img_clickpoints", bgr_to_rgb)
```

/replab_core/src/replab_core/config.py:

line 40:

```
-RGB_IMAGE_TOPIC = '/camera/color/image_raw'
-DEPTH_IMAGE_TOPIC = '/camera/depth/image_rect_raw'
-POINTCLOUD_TOPIC = '/camera/depth_registered/points'
-RGB_CAMERA_INFO_TOPIC = '/camera/color/camera_info'
```

it is recommended to study about topics and their functionality in ROS.

Camera alignment:

When using the cam_align.py script, define the correct usb port as an argument (/dev/video*). You may need to connect the host port to the Docker when invoking the docker. Do this by adding the following flags to the invocation command:

```
--device=/dev/video0 --device=/dev/video1 --device=/dev/video2
```

Proper invocation of the cam_align.py script (for example for /dev/video1):

```
python cam_align.py --cameraA /dev/video1 --ref_image <ref_image>.jpg
```

Control noise:

Calibration script is found at:

/root/ros_ws/src/replab/replab_core/src/replab_core/compute_control_noise.py

Change Line 47:

```
x, _, _, _ = np.linalg.lstsq(targets, actual, rcond=-1)
```

Training data:

Training data has corrupted files.

We deleted them and added code to skip the deleted files.

48527.npy

14930.npy

Also, please note that Replab did not provide the optimizer state of the pretrained models, so continuing training those models based on data that we collected is not ideal.

Calibration

As entailed by the original guide, there are two calibration steps:

1. Control noise - this must be done, due to assembly and manufacturing variance.

2. Robot-Camera calibration:

If using the d435 camera, we recommend using our reference image under the path:

/root/ros_ws/src/replab/replab_core/scripts/reference_020920.jpg.

If using a different camera setup or wanting to recalibrate the camera-robot matrix, follow these instructions:

In addition to the process described in the article, use our new defined function in the controller.py: `calibration_positions()`

This function will move the arm to 15 predefined, uniformly distributed coordinates in the arena, with the end effector of the arm facing downwards. At each point the robot will pause and wait for command to continue. Click on the clickWindow.py GUI on the end effector, save the point and then continue to the next point.

Results analysis

Algorithm: Random xyz-theta
Method: grasp without returns to the arena.
Experiment results: 10/60

Algorithm: Random theta
Method: grasp without returns to the arena.
Experiment results: 16/60

Algorithm: Principal-axis
Method: grasp without returns to the arena.
Experiment results: 16/60

Algorithm: Fullimage, Replab pretrained net.
Method: grasp without returning to the arena.
Experiment results: 18/60

Algorithm: Pinto2016, Replab pretrained net
Method: grasp without returns to the arena.
Experiment results: 9/60

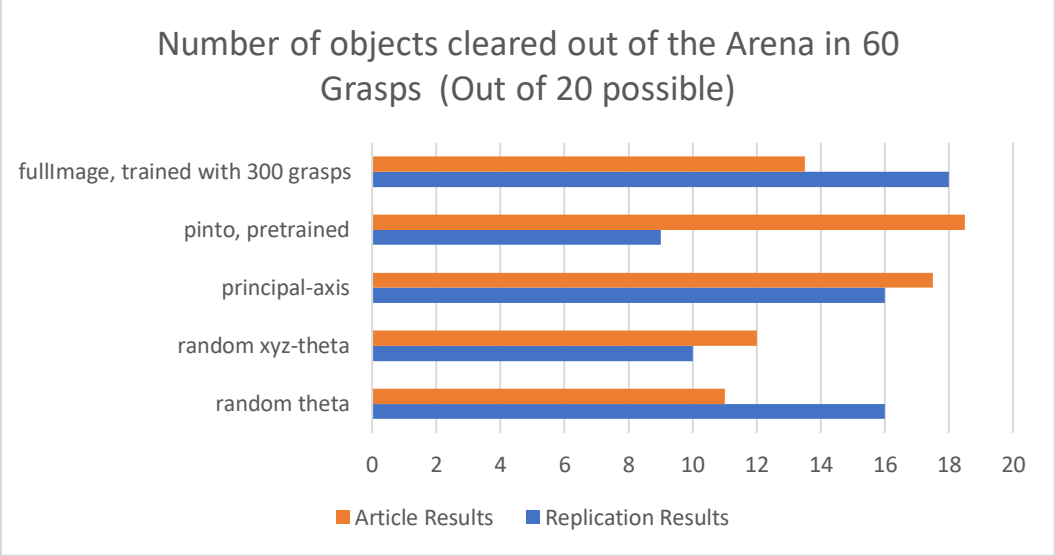

Figure 1 - our setup performance vs the article's

Conclusions

We will compare and evaluate performance in two ways:

1. Our environment versus the article's
2. Learning algorithms versus non-learning algorithms

As we can see from figure 1, the article's performance is similar to our performance. The Fullimage method grasped more objects in our setup but Pinto (cropped image) grasped less then portrayed by the Replab article. Our hypothesis is that the Pinto algorithm's poor performance is linked to the change of camera. The Realsense D435 camera has higher resolution Than the original SR300 Camera. Pinto algorithm uses a cropped image to evaluate the probability of success on a grasp, and the image crop is done with constant radius of 48 pixels. That crop radius was derived specifically for the SR300 camera. The D435 high resolution causes the crop procedure to yield a smaller filed of view around the targeted object. That can be seen in the following figures (6-8):

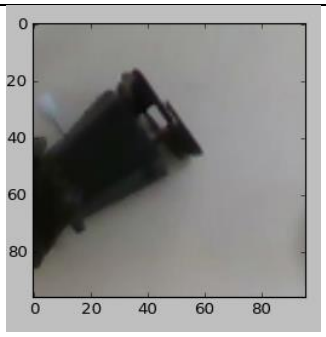

Figure 6

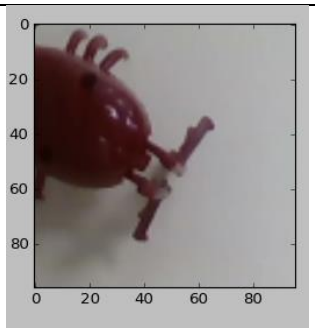

Figure 7

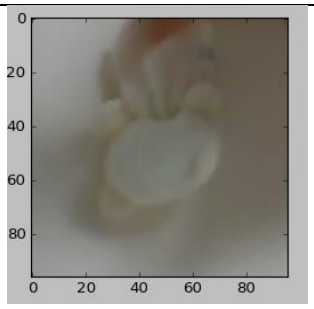

Figure 8

Figure 9

We can see that in our input images the object covers a larger part of the image in comparison to Replab's input images (figure 9). We propose that this phenomenon causes the poor performance of the algorithm, as the input images fed to the network are inherently different than the ones the network was trained on.

When comparing the non-learning algorithms to the learning Fullimage we can see that Fullimage is marginally better than the non-learnings. This resembles the article's results and led us to the conclusion that the setup was reproduced, and that training algorithms based on data collected by different cells is possible and does improve the algorithms performance on all cells.