Nvidia Xavier Manual

How To Run The Car?

1. Easy way (Automated)

- 2 Buttons on the desktop : **EPIC** and **ROSCORE**
- First click on the **ROSCORE** button, which will then start the roscore on a terminal
- Then click on the **EPIC** button, which will start the gui_epic_screens.py script and start the GUI from the path /home/xavier1/catkin_ws/src/gui/src
- Choose manual mode for joystick control, and autonomous mode for complete autonomous function. It will start the shell scripts for the given function in the background
- Then on the GUI you can choose the detection function such as action recognition or free floor detection etc.

2. Other way (Manual)

- Start on a separate terminal :
  `roscore`
- On another terminal:
  `cd catkin_ws`

  `catkin_make`

  `source devel/setup.bash`

  `rosrun gui gui_epic_screens.py` (to start the GUI)
- If any problems occur during the start of the ROS process, please check the capital for ROS.
  OR
- start the shell scripts separately for autonomous/manual function
  For autonomous :
  `roslaunch human_stalker follow_person.launch`

  `roslaunch human_stalker follow_person.launch`

  `roslaunch epic_car_description create_model.launch`

  `roslaunch realsense2_camera rs_camera.launch *`

  `rosrun human_stalker image_resize.py*` Launch Camera Image Resizer & Compressor

  `rosrun image_transport republish raw in:=image_resized out:=/image_compressed*`

`roslaunch epic_drive_functions cartographer.launch` (Launch Cartographer)

`rosrun obstacle_avoidance info_uss.py` (Launch Obstace Avoidance (Utraschall Sensoren)

For manual :

`roslaunch epic_drive_functions joystick_teleop.launch`

`roslaunch taraxl_ros_package taraxl.launch`

`rosrun human_stalker image_resize.py*` Launch Camera Image Resizer & Compressor

`rosrun image_transport republish raw in:=image_resized out:=/image_compressed*`

`roslaunch epic_drive_functions cartographer.launch` (Launch Cartographer)

- Then start the action detection script
  `rosrun gui run_ehpi_fast.py`

* : not neccessary for local processing, **only** for image transfer to Trainings PC

Action Recognition Script in Detail

run_ehpi_fast.py

Main

Image_size should represent the dimensions of the current camera or the video.
Use_action_recognition = True means that action recognition will be used, could also set to False, which means that only Pose Estimation will be used and the hip information will be published for all human inputs. Makes sense to use if only single person is in the scene
INPUT PROVIDER :

```
######################### INPUT PROVIDER ##########################
# input_provider = VideoDirProvider(camera_number='/home/xavier1/catkin_ws/src/gui/src/Office.mp4',image_size=image_size, fps=fps)
# input_provider = WebcamProvider(camera_number=0, image_size=image_size, fps=fps)
input_provider = WebcamProvider_USBCAM(camera_number=2, image_size=image_size, fps=fps)
# input_provider = ImgDirProvider(img_dir ="/Users/sk82620/Pictures/Camera Roll/yolo",image_size=image_size, fps=fps)
```

- **VideoDirProvider :** Give the video path to be processed
- **WebcamProvider :** ONLY for RealSense camera, with pipeline code
- **WebcamProvider_USBCAM :** For all USB cameras, including RealSense (no special libraries are in this case needed, it will be taken into consideration as normal USB camera)
- IF “Can’t capture frames” occurs, change the camera_number to 0-1!

For **saving the processed video** :
If desired, the processed video could be saved under the given path with desired “fourcc” rate → 25-30 for real time, could be saved slower or faster.
Webcam videos CANNOT be saved, only the given videos!
There is a one last function at the end of the script (*out.write(img)*) which is needed to save the videos.

Action Recognition Model

From ehpi_action_recognition.config import ehpi_model_custom_4_ADAM

- The path to the trained model should be defined in the config.py script and should be imported in the beginning of the main run_ehpi script.

```
def get_stabelized_action_recognition(human_id: str, action_probabilities: np.ndarray,joints):
    queue_size = 20
    if human_id not in action_save:
        action_save[human_id] = []
        action_save[human_id].append(deque([0] * queue_size, maxlen=queue_size))
        action_save[human_id].append(deque([0] * queue_size, maxlen=queue_size))
        action_save[human_id].append(deque([0] * queue_size, maxlen=queue_size))
        action_save[human_id].append(deque([0] * queue_size, maxlen=queue_size))


    argmax = np.argmax(action_probabilities)
    for i in range(0, 4):
        if i == argmax:
            action_save[human_id][i].append(1)
        else:
            action_save[human_id][i].append(0)

    # print(sum(action_save[human_id][0]) / queue_size) ----gives the probabilities

    WALK = float("{:.2f}".format((sum(action_save[human_id][1]) / queue_size)))

    if (joints[0][0] < 0.50)and(joints[1][0] < 0.50)and(joints[2][0] < 0.50)and(joints[3][0] < 0.50):
        WALK = 0 #  setting the probabilty of walk to zero if the feet and knee joints are not visible

    return [sum(action_save[human_id][0]) / queue_size,
            WALK,
            sum(action_save[human_id][2]) / queue_size,
            sum(action_save[human_id][3]) / queue_size]
```

- Config file contains all the necessary weights for the models (YOLOV3, Pose Estimation and ShuffleNetV2)
- Copy the new trained model in the models folder under : ehpi_action_recognition\data\models

From nobos_commons.data_structures.humans_metadata.action_4_class import Action

- In the action.py script you will see the Action class which consist of lists of the trained actions. There might be more lists commented, they were used for previous applications. It is important the the current list should contain the new actions for the model (with the right sequence as the training label →Example: if WALK has the label 0, then it should be the first action name in the list) The same sequence should be given in the **return** as seen in the figure above.
- The number of actions is import to determine some properties. For example for 4 actions
  - Append function will be done for 4 times.
  - For loop should also go through (0, total action number) so that each action will get its probability while the program runs.
  - Function should also return sum(action_save…) list with 4 elements. (3)

Action Network

- Trained and imported action recognition model should then be loaded in “Action Network”. Parameter of ShufflenetV2 should also be adjusted, if number of action classes (n_class) and input_size has been changed during training.
- Input_size is mostly fixed, asit is equal to the EHPI length of 32 (look at the paper “Simple Yet Efficient Real-Time Pose-Based Action Recognition” for details about EHPI)
- New trained model should be then loaded (ehpi_model_custom_4_ADAM), which was imported in the beginning from config file.

```
# Action Network
action_model = ShuffleNetV2(input_size=32, n_class=4)
state_dict = torch.load(ehpi_model_custom_4_SGD)
action_model.load_state_dict(state_dict)
action_model.cuda()
action_model.eval()
feature_vec_producer = FeatureVecProducerEhpi(image_size, get_joints_func=lambda skeleton: get_joints_jhmdb(skeleton))
action_net = ActionRecNetEhpi(action_model, feature_vec_producer, image_size)
```

Pose Estimation

- Pose Estimation network will be initialized so that it could work faster in real time

```
# Pose Network
pose_model = pose_resnet.get_pose_net(pose_resnet_config)
logger.info('=> loading model from {}'.format(pose_resnet_config.model_state_file))
pose_model.load_state_dict(torch.load(pose_resnet_config.model_state_file))
pose_model = pose_model.cuda()
pose_model.eval()

with open('/home/xavier1/catkin_ws/src/gui/src/networks/pose_estimation_2d_nets/human_pose.json', 'r') as f:
    human_pose = json.load(f)

pose_net = Pose2DNetResnet(pose_model, skeleton_type,human_pose)
pose_tracker = PoseTracker(image_size=image_size, skeleton_type=skeleton_type)
```

Program Schedule

1. Each Frame will be given as input to the pose estimation network, which can detect the poses of multiple people at the same time
2. Bounding boxes (BB) are then created with the estimated joint coordinates from pose estimation
3. BB and human (joint coordinates) will be given to tracker with the list of previous human list from previous frame.
4. New humans from current frames will either get the ID from previous humans if they are similar enough or will assigned with a new ID.
5. Different than the original code, YOLOv3 is not being used due to new and better pose estimation algorithm,which can detect multiple human at the same time. Also human tracking algorithm is reduced to single function and re-detect human and additional merge functions have been left out.
6. Humans are then given to the action recognition network.

7. If HAND_WAVE is detected, human will be logged in and will get a yellow BB. Only the information (such as hip coordinate etc.) will be published so that the car would only consider the commands coming from this person or follows only him/her.
8. If the logged in person makes CAPITULATE, person will be no longer logged in and the car looks for other people to stalk.
9. ROS publisher functions are defined at the beginning of the script, which will then be called in the cases in logged in or detected functions later. The needed publishers could be called as desired or commented if not needed.

Action Recognition Training Script in Detail

Main Requirements :

Pre-processing

- In order to train action recognition model, dataset should first preprocessed. The model expects an input of 15 Skeleton coordinates in following order ; nose, neck, hip center(belly), left shoulder, left elbow, left wrist, right shoulder, right elbow, right wrist, left hip, left knee, left ankle, right hip, right knee, right ankle.
- If a dataset already contains the skeleton coordinates (not dependent on exactly these joints or in this order) a direct preprocessing on the coordinates could be done with **skeleton_numpy.py** script.
- This script is written for NTU_RGBD dataset skeleton data preprocessing, where in the beginning the joints of the NTU dataset was matched to our 15 joints in the correct order.
- It is important to notice the joint coordinates representation of the dataset and bring them into the correct order.
- An EHPI (a single sample for the training) consists of 3 (x,y,z (z could declared as 1 or 0 while in this case we only work with 2D data, z would later be ignored during training) coordinates of 15 joints for 32 frames in a video. This makes in the end a single line with 1440 elements for a clip of 32 frames. A line is formed like the following; first frame, 3 coordinates of 15 joints in order, second frame, 3 coordinates of 15 joints in order, … , until 32 frames
- The corresponding label for this clip will then be saved in y-train/test file.
- If the dataset doesn’t contain the joint coordinates but only pure video data, the needed skeleton coordinates can be extracted **with run_ehpi_for_data_preprocessing.py** script, where each video in the dataset would go through the human detection and pose estimation algorithms and the joint coordinate outcomes will be saved in the correct format.
- **run_ehpi_for_data_preprocessing_better.py** script only needs the dataset path and the action folders in that path should be named correctly so that it would get the right action ID. For each file it will go in the folder and process all the videos inside with the given action ID from the name.
- **run_ehpi_for_data_preprocessing.py** on the other hand is not and optimized script and each folder path for actions are given separately. It could be usefull if there are many weird folders that are located in different locations. Otherwise **_better.py** script is suggested to be used.

```
for class_name in glob.glob('/home/xavier1/DATASET/*'):
    print('CLASS NAME : ', class_name)

    if (class_name.find('WAVE') != -1):
        action_ID = 0
    elif (class_name.find('WALK') != -1):
        action_ID = 1
    elif (class_name.find('IDLE') != -1):
        action_ID = 2
    elif (class_name.find('CAPI') != -1):
        action_ID = 3

    str_class_name = str(class_name)
    for name in glob.glob(class_name + '/*'):
        cap = cv2.VideoCapture(name)
```

- It is important to notice that each video in the dataset should have a single label (should contain a single action and if possible single person) so that the labeling process can go smoothly.
- Train and test data split is decided through a, meaning that 20% of the data will be saved in the test file and rest in train. The rate could be changed easily as desired by changing the a division number.
- If no split is desired, complete data could be saved in the “complete_X_train” file.
- Preprocessed own dataset is saved in **/home/xavier1/PrePro** and could be used in training experiments if desired.

Training Script

- The folder containing the preprocessed X_train, y_train (if exists X_test and y_test) should be copied in the path : **\ehpi_action_recognition\data\datasets**
- Later on the name of the folder will be called in the EhpiDataset Class, where each sample (line with the length of 1440) will be transformed in to a matrix form of 32,15,3. This will be the

training and test set.

```python
def get_train_set(dataset_path: str, image_size: ImageSize):

    num_joints = 15
    left_indexes: List[int] = [3, 4, 5, 9, 10, 11]
    right_indexes: List[int] = [6, 7, 8, 12, 13, 14]
    print("I am here")

    # NTU_skeleton_preprocessed_selected
    # NTU_skeleton_preprocessed_NAN_gone_no_Aug
    #NTU_skeleton_preprocessed
    #2019_03_13_Freilichtmuseum_30FPS
    return EhpiDataset(os.path.join(dataset_path, "NTU_NEW_4_Class_with_IDLE"),
                       transform=transforms.Compose([
                           RemoveJointsOutsideImgEhpi(image_size),
                           # RemoveJointsEhpi(indexes_to_remove=foot_indexes, indexes_to_remove_2=knee_indexes, probability=0.25),
                           ScaleEhpi(image_size),
                           TranslateEhpi(image_size),
                           FlipEhpi(left_indexes=left_indexes, right_indexes=right_indexes),
                           NormalizeEhpi(image_size),
                           GaussianNoise(image_size)
                       ]), num_joints=num_joints, dataset_part=DatasetPart.TRAIN)


def get_test_set(dataset_path: str, image_size: ImageSize):
    num_joints = 15
    return EhpiDataset(os.path.join(dataset_path, "NTU_NEW_4_Class_with_IDLE"),
                       transform=transforms.Compose([
                           RemoveJointsOutsideImgEhpi(image_size),
                           NormalizeEhpi(image_size)
                       ]), dataset_part=DatasetPart.TEST, num_joints=num_joints)
```

- EhpiDataset Class has some functions implemented. Some of these functions are mandatory
  - RemoveJointsOutsideImgEhpi
  - ScaleEhpi
  - TranslateEhpi
  - NormalizeEhpi

  to maintain the integrity of the set, while some of them are arbitrary
  - FlipEhpi : to flip the right and left arm joints so that left and right side wouldn't create a difference during action recognition
  - GaussianNoise : to add a little Gaussian noise on the coordinates in order to inhibit over training of the data
- It is again important to be careful about the image size. Test and train sets should share a common image size.
- In train_loader : shuffle = True and sampler = sampler parameters can't be used at the same time. If the dataset has almost equally distributed data but the classes are saved as clusters (first all the samples with the label 0, then all the samples with the label 1…), shuffle = True parameter should be used so that the samples are chosen randomly during training. If classes are **not** equally distributed, then **sampler** parameter should be used so that class with more samples would **not** have extra advantage compare to other classes with less samples.
- Trained model will be saved in the folder with the desired name given in the model_path, which will be created in the path **\ehpi_action_recognition\data\models**
- Each of the trained weights will be saved as .pth data in that folder. The folder name under which it will be saved and the names of each .pth file is predefined under #config part, where the first row is for file name and second one for each .pth file.
- .pth files in the same folder will be later separated through their epoch_checkpoints.

- After defining the parameters for training (such as weight decay, learning rate, batch size etc.) check again the ShuffleNetV2 parameters (n_class) so that it is equal to the number of classes you want to train it with.
- The main action in training script happens actually in **TrainerEhpi** class

```
if __name__ == '__main__':
    seed = 0
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)

    # TEST DATASET
    test_set = get_test_set(ehpi_dataset_path, ImageSize(1920, 1080))
    test_set.print_label_statistics()
    test_loader = DataLoader(test_set, batch_size=32, shuffle=True)

    # TRAIN DATASET
    parameters = dict(
        lr = [0.01]
        ,batch_size = [64])
    param_values = [v for v in parameters.values()]
    # mit ADAM optimizer -- normal basic NTU_60_csv with shufflenet_v2 , SHUFFLE TRUE

    for lr, batch_size in product(*param_values):
        print(lr,batch_size)
        train_set = get_train_set(ehpi_dataset_path, image_size=ImageSize(1920, 1080))

        sampler = ImbalancedDatasetSampler(train_set, dataset_type=EhpiDataset)
        print("sampler is also done")
        train_loader = DataLoader(train_set, batch_size=batch_size, num_workers=1, shuffle=True)
        #  sampler=sampler,
        print("train_loader is done")

        # config
        model_path=os.path.join(models_dir, "NTU_NEW_4_Class_TRAIN_ONLY_WITH_IDLE_SGD_151")
        train_config = TrainingConfigBase("NTU_NEW_ACTIONS_{}_{}".format(lr,batch_size),model_path)
        print("model name/folder at least should be created")
        train_config.weight_decay = 5e-4
        train_config.num_epochs = 151
        train_config.learning_rate = lr
        train_config.batch_size = batch_size

        train_config.learning_rate_scheduler = LearningRateSchedulerStepwise(lr_decay=0.1, lr_decay_epoch=40)
        # not in normal train script but could be useful to improve the learning rate

        print("starting trainer ehpi")
        trainer = TrainerEhpi()

        losses_out, accuracies_out_seq ,losses_out_test,accuracies_out = trainer.train(train_loader, train_config, model=ShuffleNetV2(input_size=32, n_class=4), test_loader = tesloader)
```

Trainer_EHPI

- Optimizer could be easy changed from SGD to Adam etc. Important is paying attention to the parameters which optimizer takes as input (such as weight decay)
- With tensorboard extension, graphs for losses (such as train and validation) or accuracy in validation set can be saved per epoch. They will be saved in the “runs” order in the current project file (if doesn’t exist it will be automatically created).
- Must be careful with accuracy types → the calculated accuracy in trainer_ehpi files are clip accuracies: a label for each 32-framed clip will be predicted and compared with the ground truth label for the given batch size and the mean of all will be calculated.
- If the “real” accuracy (not for each clips in the video but for the video itself) is asked, then the trainer_ehpi script should be chosen carefully. Clips in the same video clip will be given to prediction one after each other and the mean of the labels (with its uncertainty score) will be calculated to a single label for the clip, which will be then in the end compared with the ground truth label of the video. This process is not exactly the video accuracy, but clip-based video accuracy.
- Another possibility would be to calculate video accuracy directly with original video length and not using clips at all. For this approach the model should be trained as an LSTM model and the model length should vary with the given video length.

Possible ROS issues

1. Setup is missing

If ROS setup is not defined in the bashrc file, it may require a setup step each time you open a terminal

```
source /opt/ros/melodic/setup.bash
```

Then you should be able to run roscore if there is no IP problem

2. IP problem

To change MASTER URI if network settings are changed:

```
sudo nano /etc/hosts
```

IF connected with Wlan:

```
ifconfig
```

All the connections will be listed, there look under wlan0 “inet” and copy the IP adress. Paste it in the hosts file with the name jetson-xavier-1 (could be 2 is you are on the second Xavier)

```
wlan0 --> inet copy --> paste in hosts file with the name jetson-xavier-1
```

** Don't forget to delete/comment out the old IDs **
SAVE hosts file and exit
Now in terminal

```
ping jetson-xavier-1
```

if successfull you can start the roscore

you might still need to export MASTER URI and ROS IP especially if you are trying to communicate with other devices

```
export ROS_MASTER_URI = http://10.44.8.158:11311
```

```
export ROS_IP=192.168.1.1
```

3. Python problem

These packages should be installed to use ROS with python3 :

```
sudo apt-get install python3-pip python3-yaml
```

```
sudo pip3 install rospkg catkin_pkg
```

And the main script to execute should have the following code on the first line

```
#!/usr/bin/env python2.7
```

If it says it cannot find such a script, make the script executable by first going into folder where the script in located in the terminal and then execute the command

```
chmod +x name_of_the_script.py
```

Following commands should already be in the bashrc file and executed automatically in every new start, if any probblems check it or add new commands
Illegal instruction (core dumped) -- due to numpy 1.19.5

```
export OPENBLAS_CORETYPE=ARMV8
```

To use pyrealsense2

```
export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python3.6/pyrealsense2
```

To add commands to be executed during new start, add the commands to bashrc file
To open .bashrc file, execute the command:

```
code ~/.bashrc
```