SAILing with Loomo

Visual Intelligence for Transportation (VITA)
École Polytechnique Fédérale de Lausanne (EPFL)

March 1, 2022

1 Overview

Urban mobility is on the brink of an intelligent revolution thanks to the resurgence of deep learning that has been rapidly advancing many building blocks for autonomous vehicles, *e.g.*, object detection, semantic segmentation, scene understanding, motion planning. Yet, building a mobile platform capable of acting in dense dynamic environments in a human-like manner remains a grand challenge. To attain the Socially-aware AI for last-mile mobility (SAIL), we need algorithms that can not only robustly perceive but also consciously predict and plan ahead of time.

In this course project, we will explore how to build these algorithms through an object following task in the real-world with a Segway robot named Loomo. Our goal is to build enable the robot to recognise and follow a moving object in a dynamic environment with time efficiency. We will build upon the *OpenSAIL* library and develop and tailor our own deep learning models with the techniques learned in lectures. In the first phase, we will start from public datasets and train a detection model to recognize the object of your choice. In the second phase, we will validate and further improve our models on Loomo for the following task in the real-world. To SAIL with Loomo as fast as possible, we may consider multiple options, ranging from enhancing the robustness of our detector in various conditions to building additional tracking or prediction algorithms. We will have our algorithms competing against each other through a robot race at the end of the project.

Figure 1: Loomo race at EPFL in 2019

2 Training

In most machine learning projects, a data scientist starts by using popular architectures and modifying it for his own task and data. In this project, we will be modifying a human pose estimator to perform object detection. Once a good object detector is obtained, it will be used to make our robot, Loomo, track a specific object moving from one point to another in the shortest time possible.

2.1 Milestones

Download the zip file corresponding to each milestone and fill the required missing code. The 'pass' statements in the python files indicate that it should be replaced by your own block of code. Feel free to create new files, classes, and functions to help you in your tasks. Make sure that the main scripts provided in each milestone can be called properly.

2.1.1 Milestone 1 - Implementing a COCO Dataset Loader for OpenPifPaf

OpenPifPaf[1] is our new bottom-up method for multi-person 2D human pose estimation that is particularly well suited for urban mobility such as self-driving cars and delivery robots. The underlying concept that makes OpenPifPaf a strong human pose estimator can be used for other tasks such as detection. This can be done by writing a proper dataset loader specific for detection, the main task for the first milestone.

We will implement a dataset loader, similar to CocoKeypoints[2], that takes the full COCO dataset and prepares it to be used for detection by OpenPifPaf. The dataset loader requires the use of 'pycocotools' to read the images and annotations, the combination of both bounding box annotations and class labels to create new ground-truth labels, and the proper preprocessing of the images for OpenPifPaf. The new ground-truth labels are created such that there are the same number of PIFs as there are object class labels, and each PIF represents the center of the bounding box of the object.

Note. We will not be using the same annotations as OpenPifPaf. We will be using annotations that contain all the COCO classes with their bounding boxes.

Note2. Currently in OpenPifPaf, the PIFs represent the 17 keypoints. Even though the new ground-truth labels are not keypoints, add these labels with a 'keypoints' key in the final annotations dictionary. This will be clearer once you understand OpenPifPaf and the dataset it uses.

To setup your environment:

1. Install OpenPifPaf by running:

```
pip3 install Cython numpy
pip3 install 'openpifpaf[train,test]'
```

[2]https://github.com/vita-epfl/openpifpaf/blob/master/openpifpaf/datasets.py

2. Follow this file[3] to download the MSCOCO data. Make sure the data-mscoco folder is located in the same directory as your training script.

3. Create a directory called 'outputs' in the same directory as your training script.

In this milestone, the 'train.py' and 'predict.py' run with the same commands in the official OpenPifPaf repo (**openpifpaf.train** and **openpifpaf.predict**) and changing the 'headnets' argument to '--headnets pif80' indicating 80 COCO class labels.

- **Example command to train:**

```
python train.py --lr=1e-3 --momentum=0.95 --epochs=10 --lr-decay 60 70
↪ --batch-size=8 --basenet=resnet50block5 --head-quad=1 --headnets
↪ pif80 --square-edge=401 --regression-loss=laplace --lambdas 1 1 1
↪ --auto-tune-mtl
```

- **Example command to predict:**

```
python predict.py --checkpoint <location of trained model> <image to
↪ test> --show --pif-fixed-scale=1.0 --instance-threshold=0.0
```

2.1.2 Milestone 2 - Implementing a Custom Dataset Loader for OpenPifPaf

Since this project requires a robot to follow a specific object, it is important to make sure that your model is a perfect detector for this object rather than having a detector that is good for many objects. One way to do this is to prepare your own dataset which can be combinations of different dataset, images you collected, or etc. In this milestone, you will specify an object that you want your model to focus on, prepare a dataset, and then create a custom dataset loader and predictor.

2.1.3 Milestone 3 - Improving OpenPifPaf Detections

After preparing a dataset and a good detector, the next step is to improve the model's performance. This is where you will be using your background knowledge and information from class to improve your detections. This involves hyper-parameter tuning, data augmentations for both training and prediction, and etc.

2.1.4 Milestone 4 - Handling Model Failures (Optional)

Although this is an optional milestone, it is rather an important one since it can differentiate your project from other team's project. In this milestone, you will need to handle the different failures of your network such as misdetections. There are many ways to solve these challenges, and this will give your model a good competitive edge.

[3] https://github.com/vita-epfl/openpifpaf/blob/master/docs/datasets.md

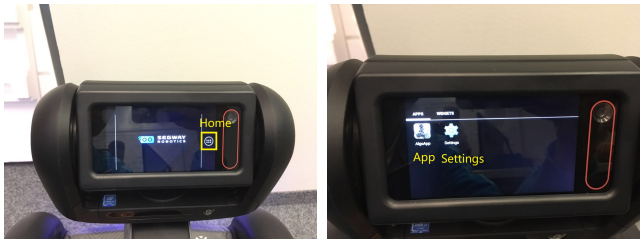
3 Deployment

Once a detection model is well-trained, you can now deploy it to Loomo, a Segway robot, for the object following task. To ease the model deployment, we will use a communication protocol between a robot and a cloud computer, *e.g.*, your laptop or a GPU workstation. With this protocol, you can run a trained detector on the cloud computer that receives image streams from the robot and returns results to it.

3.1 Robot

3.1.1 Turn on your robot

To turn on a Loomo, press the button on its body and the button on its head in turn. Figure 2 shows the main pages after you turn on the robot.

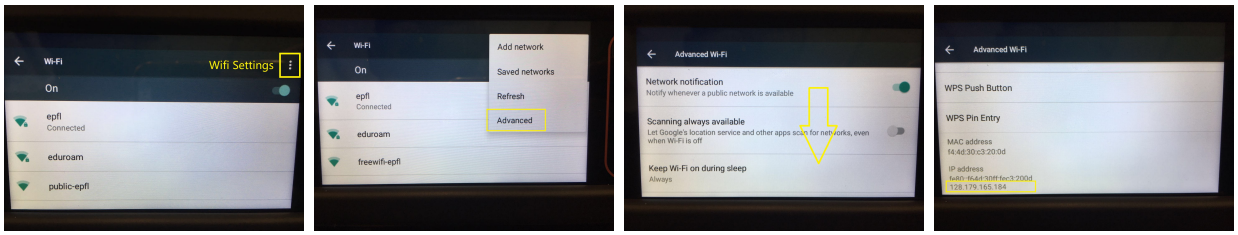


(a) Home **(b)** Apps

Figure 2: Turn on robots

3.1.2 Find dynamic IP address

To establish the communication between the robot and your pc via Wi-Fi, you need to find the IP address, as shown in Figure 3.

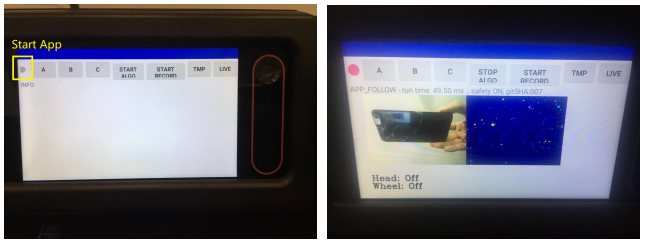


(a) Step 1 **(b)** Step 2 **(c)** Step 3 **(d)** Step 4

Figure 3: Find IP address

3.1.3 Start robot app

You can now get back to the main page by tapping the robot's ear. Figure 4 shows how to start the robot app for object detection.



(a) Before Start **(b)** After Start

Figure 4: Start App

3.1.4 Turn on controllers

Once the app is initialized, you can switch on the controller of the robot's head and wheel with the button

- A: head control
- B: wheel control

3.2 Cloud

Once the robot is settled, you can now start your model on the cloud.

1. Test the IP address and the video stream

   ```
   python test.py --ip-address <robot ip>
   ```

2. Run your model

   ```
   python client.py --ip-address <robot ip> --checkpoint
   ↪  <location of trained model>
   ```

3.3 Tuning

To improve the performance of your tracking robot, you may want to fine-tune the resolution of your image stream as well as the parameters of your controllers. A eash way to change these parameters is to push a configuration file into the robot.

1. Connect your laptop to a robot through WiFi

   ```
   add connect <robot ip>
   ```

2. Push configuration file

   ```
   adb push follow.cfg /sdcard/
   ```

The default parameters are as follows:

```
1.50    % k_p, head controller
0.10    % k_i, head controller
0.01    % k_d, head controller
0.50    % k_p, vehicle controller
0.01    % k_i, vehicle controller
0.01    % k_d, vehicle controller
8.00    % image downscale rate, the original image is 640 x
↪  480
```

References

[1] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. Pifpaf: Composite fields for human pose estimation. *CVPR, arXiv preprint arXiv:1903.06593*, 2019.

4 Appendix: Environment Setup on Robots

Prior to the model deployment, some preparations on robots are required to setup the hardware and software environments. First, the robot needs to be switched to the *developer* mode, as shown in Figure 5.

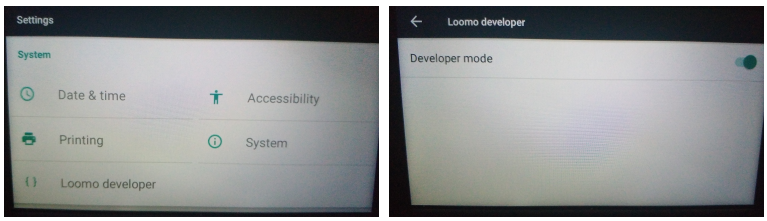


Figure 5: Loomo Developer Mode

Furthermore, an android app designed for this project needs to be installed. The source code can be found at https://github.com/segway-robotics/loomo-algodev. It is recommended to use Android Studio 3.1.4. to build and install the app.

Before compiling the source code, make sure that the local paths to ndk and sdk are added in the *local.properties* file under *gradle*, as shown in Figure 6.

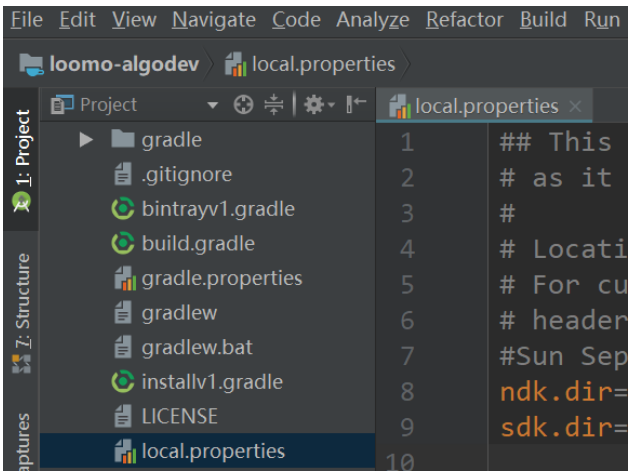


Figure 6: Gradle settings

To build the source code, open the file *AlgoFollow.cpp*, and build the module as shown in Figure 7.

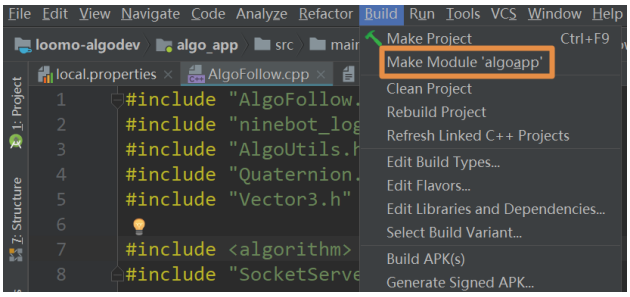


Figure 7: Build project

Once the source code is built successfully, we can now install it to the robot, as shown in Figure 8. Note that our laptop needs to be connected to the robot through WiFi, as described in Section 3.3.

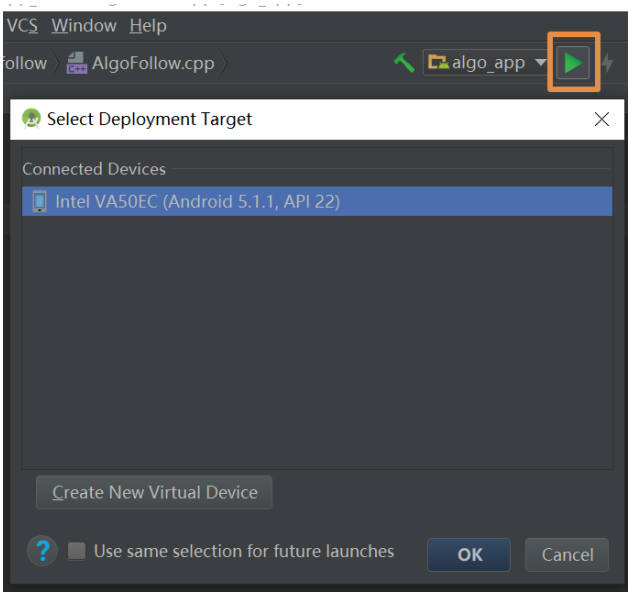


Figure 8: Install app