EE492 Senior Project

Human Trajectory Forecasting

Mustafa Büke

Department of Electrical and Electronic Engineering

Project Advisor: H. Işıl Bozma

July 1, 2024

Boğaziçi University

Bebek, Istanbul 34342

Abstract

Multi Object Tracking (MOT) is a problem that focuses on dynamic object(s) in a given video sequence. With the improvements in GPUs and the rise of the Deep Learning, the methods can deal with this problem very accurately. In this project, different tracking algorithms are tested and compared. One of the algorithms, UCMC [1], is chosen as the tracker and the code for this algorithm has been used. There were some bugs inside the code. Those bugs have been fixed and the algorithm is run on Jetson development kits.

Contents

List of Figures

List of Tables

1 Introduction

Many robotic tasks including those associated with crowd management, autonomous navigation, and human-robot interaction. require predicting the trajectory of human movements. For example, in autonomous navigation, the mobile robot is required to pay extra attention to maintain social distance to the humans around it. This is a challenging problem - especially in scenarios in which the robot also is moving. As the human targets can drift considerably, tracking can become erroneous.

2 Problem Statement

The goal of this project is to forecast the trajectory of a human or a group of humans given their past movements and environmental factors. The sensory input will be RGB-D and or LIDAR data as acquired by possibly a mobile robot. The problem requires collecting data on human movements, identifying humans and developing predictive models that can anticipate future trajectories accurately.

amsmath

3 Approach and Methodology

This project consists of four stages.

- The first stage is to detect and track the objects in the scene.
- The second stage is to determine offline datasets that can be used for evaluating the performance of different methods.
- In the last part of the project, the objects are assumed to have constant velocity and the estimations are based on this assumption.
- The final part is real-time testing on a stationary and moving mobile robot.

3.1 Multi-Object Detection & Tracking

It is observed that a multitude of t tracking algorithms have evolved over time [4]. These methods are typically composed of two stages:

- In the first stage, the objects that are intended to track are detected.
- In the second stage, those objects are matched with the tracked objects.

The rest of this section presents the algorithms that have been considered in this project:

- GOTURN
- KCF
- MIL
- TSDM
- UCMC

3.1.1 GOTURN

GOTURN is a well-known tracker algorithm. It is a learning based end-to-end siamese tracker[2]. This tracker uses a simple feedforward neural network and does not require online training. The tracker learns the general relationship between object movement and appearance and can be used to track new objects. This network takes two input images and gives a prediction of the object that has been tracking. In addition, this algorithm uses RGB input images and it is not capable of tracking multi objects. However, GOTURN didn't work accurately in our test videos. This probably stems from that the model has not been trained properly. The main reason of failure has not been investigated since this algorithm is not appropriate for multi object tracking tasks.

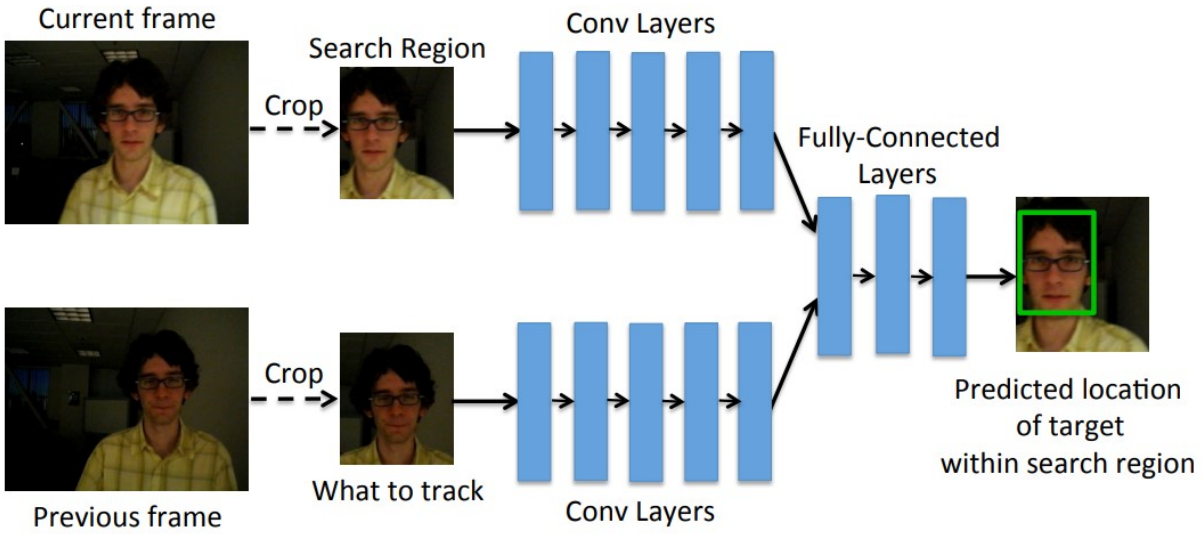


Figure 1: Tracking pipeline of GOTURN algorithm [2]

3.1.2 KCF

KCF is another OpenCV tracking algorithm[5]. This algorithm was highly successful as opposed to GOTURN algorithm. It can track the given object up to 100 FPS using CPU. Also, the tracker becomes aware when it loses the target object.

3.1.3 MIL

MIL is the last Opencv algorithm that has been tested[6]. This algorithm takes RGB images and can track a single object. Also, when the is target lost, it gives an output with highly low accuracy.

3.1.4 TSDM

TSDM is another algorithm that has been tested and this is the only one that takes RGBD images[3]. The depth information gives some clues about the background of an image. TSDM uses these clues to mask the region of interest. After masking the area, it tries to find the object of interest int the given area. It is important that the algorithm only

works on CUDA supported GPUs. The tests has been done on NVIDIA T4 GPU on Google Colab platform. The main disadvantage of this method is that it can track only one object.

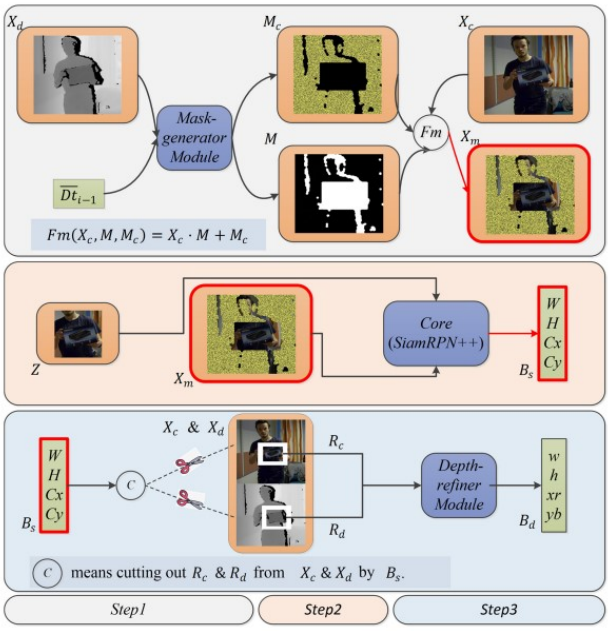


Figure 2: Tracking pipeline of TSDM algorithm [3].

3.1.5 UCMC

UCMCTrack: Multi-Object Tracking with Uniform Camera Motion Compensation is the first multi-object tracking algorithm that has been tested[1]. The input for this algorithm is the RGB image and the output gives the bounding boxes of the tracks. This algorithm uses different methods than usual tracking algorithms. For instance, the intrinsic and extrinsic camera parameters should be provided to the algorithm. Also usual intersection-over-union (iou) is not used as the distance metric.

Firstly, an object detector model is used. Then, the algorithm takes midpoints of the bottom edges of the bounding boxes as the location of the object. These midpoints are projected into the ground plane by using the intrinsic and extrinsic camera parameters to eliminate the occlusion problem and to model the objects movements more realistically. In this way, the positions of the objects in the ground plane are used.
Assume that the algorithm runs the first time. The tracker will not be used before the object detector detects the first object. When the object detector detects an object and the confidence score of the detection is high enough, the algorithm assign a track id to this object. The track status of this object will be ”tentative”. It means that this object could be a possible false alarm and it will not be necessary to track this object. When the next frame comes, the Kalman filter predicts the ground plane location of the detected object. If the object detector detects some objects, those detections are represented in ground plane as well. The locations of the objects are represented as 2-dimensional distribution functions. If the bounding box of the object is huge, the variance of the corresponding distribution function

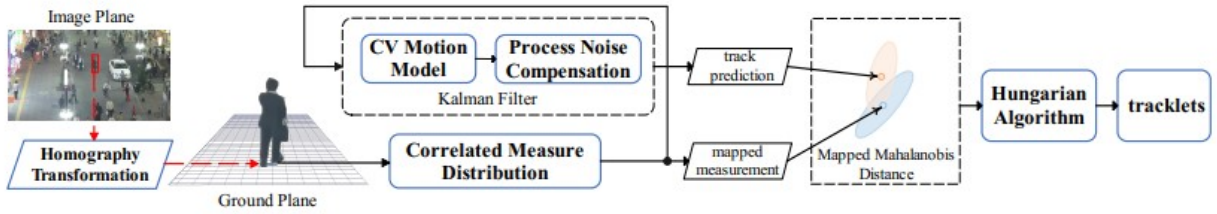


Figure 3: Tracking pipeline of UCMC algorithm [1]

will be large. Let's assume there are two detected objects in the first frame and three detected objects in the second frame. The tracker algorithm tries to match the detected objects in the first frame with the detected objects in the second frame. A cost matrix created for binary matching. The cost of every matching is measured by using Mahalanobis distance. This is distance between the distribution function of the detected object in the current frame and the predicted point in the previous frame. After building the cost matrix, Hungarian algorithm is used to optimal matching. Inside the code Jonker-Volgenant algorithm is used for this purpose. If any detection from the first frame is matched with the detection from the second frame, the track status of the object will be assigned as "confirmed", which means this detection is not a false alarm and this object will be tracked in the next frames. If an object from the first frame is not matched with the objects from the second frame, this detection will be evaluated as false alarm and the corresponding track id will not be used anymore. If any object from second frame remains after matching operation, this object will be evaluated as a tentative object. If a track has not seen for a while, t_d, the algorithm assign the status of this track as "dead", which means the algorithm will not try to match this track with new detections in the upcoming frames. This threshold value can be adjusted for specific applications. In our code, this value is equal to 100 frames. The algorithm predicts the location of a track in every new frame with the Kalman filter when it cannot match the track with a detected object.

4 Datasets

To observe the performance of the UCMC algorithm, RGBD data has been collected by using a stationary XBOX360 Kinect camera. Our dataset consists of 2 videos, each 1 minute long. One of the videos includes just one moving person and the other video includes three moving people. It was observed how successful the algorithm was in these videos. The more detailed explanation of how to form setup has been provided in Appendix B. In addition, the videos can be accessed by visiting the Github repository of this project, which is provided in Appendix C.

5 Velocity Estimation

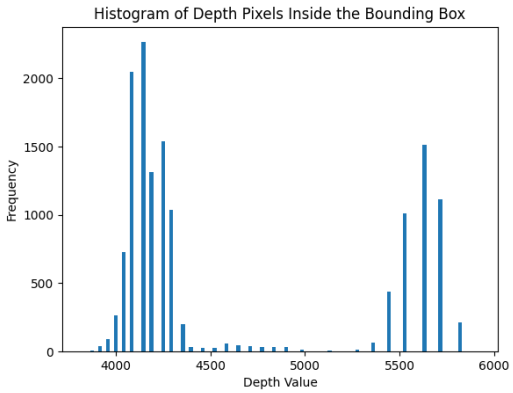
The next step was to compute the velocities of the tracks. For this, the depth data is required. One of the problem was from which point we should read the depth value. One of the approaches is to take the midpoint of the bounding box as the reference point. In most cases, this method gives good results. However, if the detector cannot draw the bounding box properly, the midpoint of the bounding box can correspond to the point on the background. This leads to misperception of the 3D position of the object. Another approach is to take the average of the depth values that are inside the bounding box. This method also have some deficits because an object like a human is not similar to a rectangle. Therefore, any bounding box that contains a human can includes the background points of the human as well as seen in Figure 5

Input: A video sequence: V, Object detector result: $Dets$, Camera parameter: $Para$
Parameter: Confidence threshold:τ, Lost time threshold:dt, The process compensation factors:σ_x, σ_y, Detection noise factor:σ_m
Output: Tracks $\mathcal{T}$ of the sequence

while frame f_k not None **do**
  $\mathcal{T} \leftarrow$ KF($\mathcal{T}, \sigma_x, \sigma_y$) {Updating the trajectory positions and distribution}
  Let $\mathcal{C} \leftarrow \emptyset$.{Initializing the cost matrix}
  $\mathcal{D}_K \leftarrow Dets(f_k)$ {Get detection result}
  for $\mathcal{D}_i$ in $\mathcal{D}_K$ **do**
    $\mathcal{Y}_i, \mathcal{R}_i \leftarrow$ Map($\mathcal{D}_i$,$Para, \sigma_m$) {Get the position and distribution of $\mathcal{D}_K$}
    for $\mathcal{T}_j$ in $\mathcal{T}$ **do**
      $\mathcal{C}_{ij} \leftarrow$ MahaD($\mathcal{Y}_i, \mathcal{R}_i, \mathcal{T}_j$)
    end for
  end for
  $\mathcal{T}_{Associate}, \mathcal{T}_{remain}, \mathcal{Y}_{remain} \leftarrow$ Hungarian($\mathcal{C}$)
  for t in $\mathcal{T}_{remain}$ **do**
    if t miss time $> dt$ **then**
      delete t
    else
      $t \leftarrow t$ miss time +1
    end if
  end for
  for y in $\mathcal{Y}_{remain}$ **do**
    if The confidence level of $y < \tau$ **then**
      delete y
    else
      Initializing a new trajectory, t
    end if
  end for
end while
return Tracks $\mathcal{T}$

Figure 4: Pseudocode of UCMC Track Algorithm [1]

(a) Bounding box of an human



(b) Histogram of the depth values that are inside the bounding box

Figure 5: Bounding box and the corresponding histogram

In order to handle this problem, a new solution has been proposed. The solution is based on the constructing the histogram of depth values inside the bounding box of each object and then determining the depth with the highest occurence. In other words, the peak of the histogram was intended to find. Then, the pixels that have the depth values equal to average depth were found. These pixels are the points that represent the object in 3D world. The first moment of those pixels are calculated to represent the object as a single point. In this way, the object is assumed it is located in this single point and the corresponding depth to this point is the average depth that has been found.
In Figure 6, you can see the results of the representing the object as a single point by using different methods. If the midpoint of the bounding box is chosen, the object is represented as blue dot. If the method that has been proposed is used, the object is represented as green dot. While the blue dot indicates the background, the green dot still lies on the object even the human is not clearly visible.

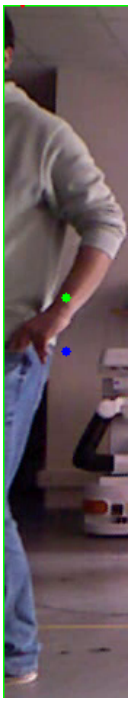


Figure 6: Different points to represent object

The last step was to obtain the world coordinates of the objects. The center point of the camera has been chosen as the origin. The x axis is along the horizontal direction, y axis is along the vertical direction, and the z axis also extends inwards from the image plane.

6 Real-Time Application

As most methods require GPU, an environment was set up on Jetson Nano development kit. The RGB-D data is collected using An XBOX 360 Kinect camera. The required steps to setup the experimental environment has been explained in Appendix A, Apendix B, and Appendix C.

7 Experimental Results

Different algorithms have been tested and compared with each other.

7.1 Comparative Study

First of all, some of the OpenCV tracker algorithms have been tested. The performance of the algorithms are tested by using AMD Ryzen 5 2500U CPU and Ubuntu 22.04 OS. TSDM was tested on Google Colab Platform using T4 GPU. The UCMC method also was tested on Jetson Nano development kit. The test videos are collected by using Kinect camera. The first algorithm tested was GOTURN. However, it was not successful in our input test images. A pre-trained Caffe model was used but it failed. Therefore, this algorithm is not a candidate tracking algorithm.

The comparison of the methods is as shown in Table I:

Method Name	**Input Type**	**Tracking per Frame**	**FPS**
GOTURN	RGB	Single Object Tracker	22
KCF	RGB	Single Object Tracker	90
MIL	RGB	Single Object Tracker	17
UCMC	RGB	Multi Object Tracker	30
TSDM	RGB-D	Single Object Tracker	10

Table I: Comparative performance

You can look at the Table I to compare the performances of the methods. As seen from the Table I, UCMC is the most appropriate algorithm for our purpose. It works very fast on Jetson Xavier. Also it is only method than can track multi objects.

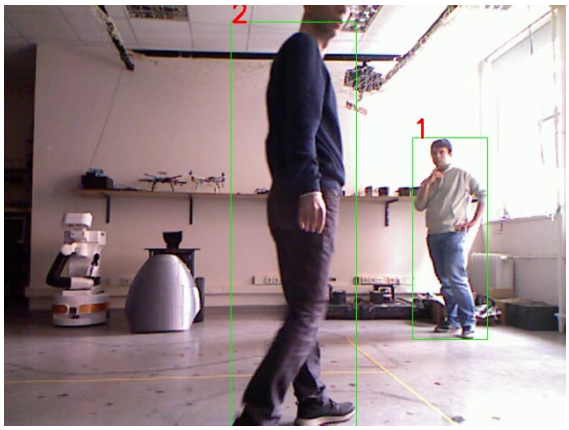
7.2 Multi-Person Velocity Estimation

The final system is setup as:

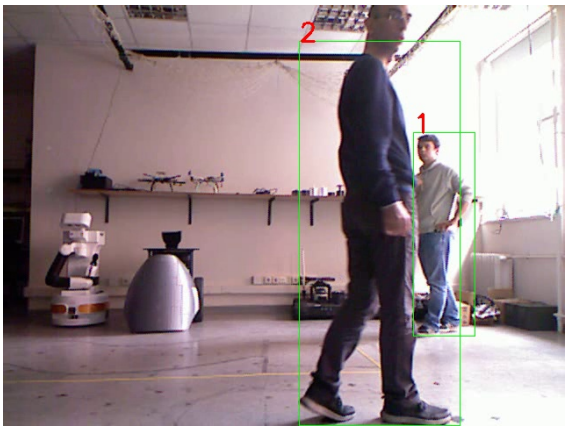
- You Only Look Once (YOLO) [7] network is responsible for detection of objects. A pretrained YOLO model has been used. The pretrained YOLO model outputs are associated with a confidence threshold τ, equal to 0.7 in our experiments, so that the detections that have lower confidence score than this threshold will not be a candidate tracks
- UCMC algorithm, which is best algorithm on MOT17 dataset according to article [1] has been used as the tracking algorithm. This algorithm uses RGB input images. Depth data has been integrated inside the algorithm to obtain the 3D motion of the objects. The velocity vector of the objects were obtained. These vectors are going to be used in order to estimate the trajectories of the object

In Figure 7, you can an example of the results. In this scene, we observe an occlusion problem, which is a common problem in tracking tasks. The algorithm has successfully tracked both of the people in the scene. This mainly stems from that the objects are matched according to their ground plane positions. Therefore, the algorithm can understand which object is in front of the other.

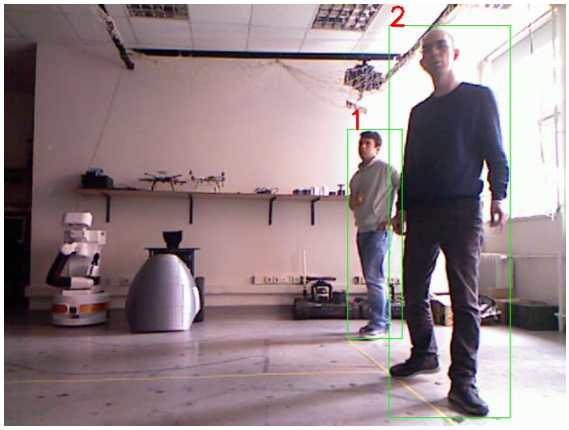
.



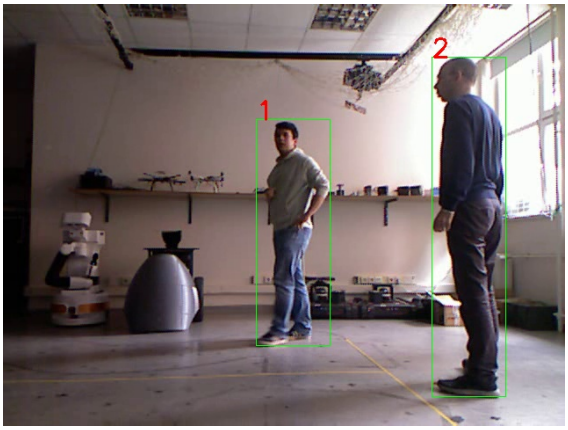
(a) First Frame



(b) Second Frame



(c) Third Frame



(d) Fourth Frame

Figure 7: An example of occlusion. The algorithm successfully handled this problem in this scene

8 Realistic Constraints

8.1 Social, Environmental and Economic Impact

As we see in most of the computer vision projects, the environmental factors are the keypoints of the problem. The camera should collect reliable data under different environmental conditions and should not be affected by any disturbances such as illuminations changes and robot's motions.

8.2 Cost Analysis

In this project, GPU plays a very crucial role. This hardware will be the most expensive part of the project. To get satisfactory performance from the models, a quality GPU should be used. The models and dataset can be used for free.

9 Conclusion

In this project, the objects can be detected by using a deep learning based method and tracked by using an algorithm. There are some points that will make the algorithm more successful. For instance, the algorithm does not take into account the classes of the detected objects. I mean, the algorithm can match a bicycle with a human. Also we can use the depth data to match the detected objects with the tracks. This can help us while dealing with the occlusion problem. In addition, a new YOLO model could be trained for specific purposes. For instance, we just want to track the humans, we could train a new YOLO model and use it for object detection.

References

[1] Kefu Yi, Kai Luo, Xiaolei Luo, Jiangui Huang, Hao Wu, Rongdong Hu, and Wei Hao. Ucmctrack: Multi-object tracking with uniform camera motion compensation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 6702–6710, 2024.

[2] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 749–765. Springer, 2016.

[3] Pengyao Zhao, Quanli Liu, Wei Wang, and Qiang Guo. Tsdm: Tracking by siamrpn++ with a depth-refiner and a mask-generator. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 670–676. IEEE, 2021.

[4] Zahra Soleimanitaleb and Mohammad Ali Keyvanrad. Single object tracking: A survey of methods, datasets, and evaluation metrics. *arXiv preprint arXiv:2201.13066*, 2022.

[5] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[6] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–990, 2009.

[7] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, January 2023.

Appendix A: How to install necessary libraries

To run the code on Jetson, you should install arhiconda. Since the Jetsons' have different CPU architecture, anaconda will not work on these devices. You can follow the instructions in the following website to install archiconda.

- Archiconda installation

After the installation, you should create a new virtual environment and activate it by using the following commands.

- conda create -n envname python=3.8
- conda activate envname

Up to this point, we could create a new virtual environment named "envname". The version of the python is highly crucial since the necessary modules are installed according to this version.
Since we want to run the code on GPU, we should install pytorch library. It is important to note that CUDA and pytorch versions should be compatible. You can get the CUDA version that is installed by using the following command.

- nvcc –version

However, there is not compiled pytorch library for Jetson that is compatible with CUDA. Therefore, you should downlaod the wheels for pytorch library. Note that you should first check JetPack version in you Jetson device by using following command:

- sudo apt-cache show nvidia-jetpack

Now you should check the compatibility matrix to install pytorch library. Note that the pytorch and JetPack versions should be compatible. Here is the related matrix.

- JetPack-Pytorch compatibility matrix

After that, you can use the following link to install the pytorch library

- Install pytorch

After the pytorch installation, the next step is to install the necessary torchvision library. If it is installed on your virtual environment you should uninstall it. You can use the following command to list the installed packages.

- conda list

You can use the following command to remove an installed package

- pip uninstall [package name]
- pip uninstall torchvision

After that you should install torchvision by using the following link.

- Install torchvision

After that you should install Ultralytics package by using the following link.

- Install Ultralytics

Finally you can install the remaining packages by using the following commands.

- pip install argparse

- pip install filterpy
- pip install scipy
- pip install lap
- pip install numpy
- pip install opencv-python

To check the installations, you can list the installed packages in your environment. To check that the pytorch library is installed with CUDA, you can create a python file named "torchcheck.py" and write the following code snippet.

- import torch
- print(torch.cuda_is_available())

Run this file and check the output. If the code print True, it means that you have installed the pytorch successfully.

Appendix B: How to install drivers for Kinect camera

This section explains how to install the drivers for Kinect camera and how to ue it in python. Firstly, you should make the connections for the camera. It is important to note that USB 2.0 is not supported in Kinect cameras. Therefore, be sure that the usb port supports USB 3.0. After making the connections, you should check that whether the Jetson has detected the camera. You can use the following command to list the connected devices to your computer.

- lsusb

When you see your camera on the list, the next step is to install the drivers. You can use the following link to install the libfreenect library.

- libfreenect installation

If you can install libfreenect successfully, the next step will be install the python wrapper. For this purpose, you can check the following website.It is highly important to note that the instructions for python wrapper is a little bit defective. For installation, you should have Cython library in your virtual environment. However, you should install the specific version of the Cython library. If it is installed, you should remove it and install it back using the following command.

- pip install Cython==0.29.27
- python wrapper for libfreenect

After completing this part, you should check the installed packages. If you can see freenect in the list, it means that you have successfully installed the libfreenect wrapper for python.

Appendix C: How to run the code

In order to run the code, you should complete the Appendix A and Appendix B successfully and activate the necessary virtual environment. After that you can clone the github repository of my work by the following command

- git clone https://github.com/islboun/3Dtrack

Inside the repository, you can type the following command to run the code. The code runs the Kinect camera and tracks the objects. Also it can print the velocities of the objects in the X and Z directions.

- python demo_rgbd.py

Also you can find the demo videos in my GitHub page.

- GitHub Repository