Karlsruhe Institute of Technology

A Real-time ROS Pipeline for Human Action Detection and Recognition

Master's Thesis of

Yonghui Jiang

Institute of Measurement and Control Systems
Karlsruhe Institute of Technology

Reviewer: Prof. Dr.-Ing. Christoph Stiller
Advisor: Jie Yu, Dipl.-Computermath.

Karlsruhe, October 2018

Declaration / Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannten Hilfsmittel selbständig angefertigt, alle benutzten Hilfsmittel vollständig angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Yonghui Jiang
Karlsruhe, 25.10.2018

Acknowledgement / Danksagung

Foremost, I would like to express my sincere thanks to my supervior Mr. Jie Yu, for offering me the chance to write my master thesis in the Computer Vision Lab in Bosch Hildesheim. His supportive guidance and advice to my master thesis motivated me and I learned a lot from him. Further, I would like to thank Prof. Stiller for offering me the mentoring privilege in MRT, KIT. At last, I appreciate all the encouragement and support from my friends and family, without them I would not have so much courage and confidence to arrive where I am standing today.

Abstract

Human action recognition has attracted an increasing amount of research attention in recent years owing to its applications in numerous domains. However, action recognition remains a challenging task due to varying viewpoints and appearance, high intra-category variation, occlusion and background clutters. In the conventional two-stream network, RGB and optical flows are leveraged to extract relevant information from appearance and temporal dynamics simultaneously. As a high-level modality, human pose has been also applied to action recognition due to the viewpoint invariance (3D pose) and robustness against appearance change and surrounding distractions. Besides, towards real world applications, development of real-time systems is also a very crucial yet challenging task.

This thesis focuses on developing a real-time application of human action detection and recognition based on human pose (both 2D and 3D) inferred from monocular RGB images and video streams. We propose a novel pipeline to integrate person detection, person tracking, pose estimation, action classification as well as real-time data pre- and post-processing (e.g. pose interpolation, visualization) into ROS (Robot Operating System) framework. Besides, we enable flexible reusage of different modules in the pipeline and provide unified APIs for module substitution and extension. Additionally, we bridge the offline training and online application of deep learning model in person detection, pose estimation and action classification. Meanwhile, we also validate effectiveness of different data processing and data structuring strategies (pose tensor) for real-time implementation. At last, we conduct experiments on JAAD dataset and evaluate the results on the merits of Intersection over Union (IoU) and run-time frame rate (in fps). The hardware platform involves Nvidia Jetson TX2 and GPU Server (Nvidia GeForce GTX 1080).

Keywords: Action detection and recognition, Robot Operating System, deep learning, real-time

Kurzfassung

Die Anerkennung menschlicher Handlungen hat in den letzten Jahren aufgrund ihrer Anwendungen in zahlreichen Bereichen zunehmend Aufmerksamkeit auf sich gezogen. Die Erkennung von Aktionen bleibt jedoch aufgrund von unterschiedlichen Gesichtspunkten und Aussehen, starker Variation innerhalb der Kategorien, Okklusion und Hintergrundgewirr eine herausfordernde Aufgabe. In einem herkömmlichen Zweistromnetz werden RGB- und optische Flüsse genutzt, um gleichzeitig relevante Informationen aus Aussehen und zeitlicher Dynamik zu extrahieren. Als High-Level-Modalität wurde die menschliche Haltung aufgrund der Invalidität des Blickpunkts (3D-Haltung) und der Robustheit auch auf die Aktionserkennung angewendet gegen Aussehensänderung und umgebende Ablenkungen. Außerdem in Richtung auf reale Anwendungen, die Entwicklung von Echtzeitsystemen ist ebenfalls eine sehr wichtige und zugleich herausfordernde Aufgabe.

Diese Dissertation konzentriert sich auf die Entwicklung einer Echtzeitanwendung zur Erkennung und Erkennung von menschlichen Handlungen basierend auf menschlicher Haltung (sowohl 2D als auch 3D), die aus monokularen RGB-Bildern und Videoströmen abgeleitet wird. Wir schlagen eine neuartige Pipeline vor, um Personenerkennung, Personenverfolgung, Posenschätzung, Aktionsklassifizierung sowie Echtzeitdaten vor und nach der Verarbeitung (z. B. Poseninterpolation, Visualisierung) in das ROS-Framework (Robot Operating System) zu integrieren. Darüber hinaus ermöglichen wir eine flexible Wiederverwendung verschiedener Module in der Pipeline und bieten einheitliche APIs für die Modulersetzung und -erweiterung. Darüber hinaus überbrücken wir das Offline-Training und die Online-Anwendung des Deep-Learning-Modells in der Personenerkennung, Posenschätzung und Maßnahmenklassifizierung. Inzwischen validieren wir auch die Wirksamkeit verschiedener Datenverarbeitungs- und atenstrukturierungsstrategien (Pose Tensor) für die Echtzeitimplementierung. Zum Schluss führen wir Experimente mit dem JAAD-Datensatz durch und werten die Ergebnisse hinsichtlich der Vorteile von Intersection over Union (IoU) und Laufzeit-Framerate (in fps) aus. Die Hardwareplattform umfasst Nvidia Jetson TX2 und GPU Server (Nvidia GeForce GTX 1080).

Contents

1. Introduction

In this chapter, we first introduce context of the research on human action detection and recognition. Then the motivation and contribution of our development are explained. At last, the organization of this thesis is described.

1.1 Action Detection, Recognition and Application

Human action detection and recognition has attracted an increasing amount of attention in recent years. The task of human action detection and recognition is to automatically analyze the ongoing activity in a given video and correctly classify the video into an action category. A video is an arbitrarily-lengthed sequence consisting of 2D image frames. In a more general case of action detection, the temporal localization is implemented to predict the starting and ending times of all occurring actions in a continuous video of a long period. In this thesis, we propose a novel pipeline for human action detection and recognition from video streams and image sequences in real-time based on the Robot Operating System (ROS) framework. we firstly focus on action recognition of well-trimmed videos or clips in which the first and the last frame roughly correspond to the beginning and end of an action, then we will also test and evaluate the pipeline on untrimmed raw videos.

Human action recognition has become an indispensable part in numerous domains and applications of computer vision. One of the most widely applied scenarios is video surveillance, which is crucial for public security, like in airports, train stations, shopping malls, banks, etc. A scene of video surveillance in an airport lobby in three distinct camera viewpoints from PETS 2007 [FT07] is illustrated in Figure 1.1.

Figure 1.1: A scene of airport surveillance in three distinct viewpoints from the PETS 2007 dataset. [FT07]

Security violations such as physical conflicts or vandalism in public places or abnormalities in the perimeter of a private house need to be detected and reported the first time by a competent action detection and recognition system.

However, understanding what a human is doing in a video sequence is never a trivial task for computers. Unlike some object detection and recognition tasks in 2D images, human action recognition remains a rather challenging problem due to several factors such as variations of view-points and appearances, complex motion styles, large intra-class diversity and distraction of background clutters. Moreover, when dealing with real-world application, effectiveness, robustness and real-time capability are taken into consideration, which increase the complexity and difficulty of such a framework together with changing and ill-conditioned scenarios.

1.2 Motivation

Bosch is working on security cameras for surveillance and they are researching on methods to recognize human actions automatically by cameras based on human pose information. Several deep-learning-based pose estimation algorithms have been developed, however there still exists a vacancy for real world application. In such a context, we are motivated to develop a competent framework to coordinate different modules and algorithms properly to achieve an effective, robust and real-time performance for the task of human action detection and recognition. The first challenge lies on the architecture design of a sophisticated framework, which is supposed to contain different components that are capable of completing the task of huamn action detection and recoginition from various input sources, and provide an unified, effective and efficient communication mechanism between the modules. Secondly, we investigate into various state-of-the-art algorithms and frameworks and make an evaluation of the utility for real-time application. Besides, we need to define the APIs for the framework, so as to realize a modular code structure and provide an easy substitution of modules as well as functionality extension. Furthermore, the transferring from offline data preparation and deep learning model training to valid online application is another challenging task we will meet.

Inspired by the work [MBM+13], which proposed a software architecture of a distributed multi-person tracking algorithm based on ROS framework, we designed a ROS-based pipeline consisting of seven main modules, i.e. *Source Provider*, *Person Detector*, *Person Tracker*, *Feature Extractor*, *Data Manager*, *Action Predictor* and *Visual Agency*. Besides we created an extra module named *Message Respoitory* to manager all the messages involved in the pipeline. We also get some inspiration from the work [Ard17], which wraps a 2d multi-person pose estimation library — OpenPose[1] into ROS nodes.

1.3 Goal Description and Contribution

Our final goal in this thesis is to develop a pipeline for the task of human action detection and recogition in a real-time fashion based on the ROS framework. An overview of the goal specifications are listed in Table 1.1. We will test on both single-person and multi-person scenario, where the former is a cropped videos or image sequences of a single pedestiran, the latter is the original video with multiple agents in the scene. The evaluation will be performed both on Nvidia Jetson TX2 and GPU Server with Nvidia GeForce GTX 1080. As listed, two metrics will be in our evaluation: the first is Intersection of Union (IoU) between the experiment action grouping results and that in the baseline, which is used to validate the effectiveness of the pipeline and will be further discussed in Section 5. The second metric we work with is the run-time performance, which is measured by frame rate in fps, where we expect the run-time performance on single agent would reach real-time and in the multi-person scenario, a semi-real-time is expected.

[1] `https://github.com/CMU-Perceptual-Computing-Lab/openpose`

Table 1.1: Thesis goal proposal.

Scenario	Testing Datasets	Hardware Platform	Accuracy (IoU)	Run-time frame rate (fps)
single-person (cropped videos)	JAAD	Nvidia Jetson TX2, GPU Server (Nvidia GeForce GTX 1080)	0.8	30
multi-person (original videos)	JAAD	Nvidia Jetson TX2, GPU Server (Nvidia GeForce GTX 1080)	0.7	10 ∼15

Contribution

In order to cover the blank spots addressed in the last section, we focus on human action detection and recognition based on human pose and build a real-time application for this purpose. The main contributions of this thesis are:

- We proposed a novel pipeline for human action detection and recognition in real-time based on the ROS framework, which integrates cutting-edge computer vision and deep learning alogirthms and frameworks related to the topic in a modular fashion.
- We provide flexible reusage of different modules and offer unified APIs for module substitution and extension.
- We validate effectiveness of strategies of data pre-processing and data structuring in datasets of scales.

1.4 Outline

Organization of remainder of this thesis is given as following. In Chapter 2, introductions of the three modalities (RGB, optical flow and pose) and Robot Operating System are provided as technical backgrounds of the thesis. The analysis of relevant literature is presented in Chapter 3 where approaches of person detection, person tracking, pose-bases action recognition as well as temporal action proposal are discussed. The pipeline architecture design and functionality explaination of different modules is presented in Chapter 4. Experiments on JAAD dataset and comparison of results are demonstrated in Chapter 5. In Chapter 6, conclusions are drawn and potential future works are presented.

2. Technical Background

In order to conduct a better presentation of this work, some fundamental knowledge in action recognition and Robot Operating System will be covered in this chapter.

2.1 Cues for Action Recognition

Action recognition can either be based on still images [ZCW+16] or videos [WXW+16b]. However, still images doesn't provide motion information that is required to predict motion-based actions. Therefore, only action recognition based on videos is discussed in this thesis. The videos provide a wide range of applications with varied actions. For these varied actions, either context information is relevant or motion information is vital or definite posture information is necessary. As RGB is meant to give context information, optical flows provide motion information and pose contributes in giving spatial posture of humans, therefore, the three cues (RGB, optical flow, pose) are essential for action recognition and they are presented here in detail.

2.1.1 RGB

Every video can be regarded as an arbitrarily-lengthed sequence of RGB frames. RGB frames (images) stack together to form a video as shown in Figure 2.1. An RGB image is a matrix with three channels where each channel has the corresponding intensity value ranging from 0 to 255 of the primary colors red, blue and green respectively. A blob of different pixels forms a structural representation of the context information contained in the scene. As convolutional neural networks have achieved great success in image-based tasks such as image classification and object detection [ZZXW18], it is comprehensible that CNN is competent in extracting effective features from image patterns.

Figure 2.1: RGB cue

2.1.2 Optical Flow

Optical flow is a canonical vision-based cue for modeling of local motion in video sequences. It describes the distribution of apparent movement or velocities of brightness patterns in an image, which is caused by relative motion between the objects and the viewer [HS81]. Given two frames at time t and time $t+\Delta t$ from the same video, an optical flow algorithm solves for a motion field where the displacement vector at every single position implies how the pixel has moved from the first frame to the second.

We can derive the equation which relates the change in image brightness at a point to the motion of the brightness pattern. We denote the image intensity by $I(x, y, t)$ as a function of the point (x, y) in the image plane and time t. Under the assumption of brightness constancy, pixel intensities are translated from one frame to the next,

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{2.1}$$

According to the Taylor series of the right side of the equation, we have,

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{2.2}$$

Then we acquire the gradient constraint equation for optical flow

$$\nabla I(\mathbf{x}, t) \cdot \mathbf{u} + I_t(\mathbf{x}, t) = 0 \tag{2.3}$$

where $\nabla I = (I_x, I_y)$ denotes the image gradient functions along the x and y direction, as shown in Figure 2.2 and Figure 2.3.

Figure 2.2: Optical flow cue in x direction

Figure 2.3: Optical flow in y direction

$\mathbf{x} = (x, y)^\tau$ is the pixel position, I_t is the temporal partial derivative and $\mathbf{u} = (\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})^\tau$ denotes the 2D velocity, which is the desired optical flow field. Apparently, the gradient constraint is an equation of two unknowns and cannot be solved without further constraint.

By assuming that the pixels in a neighborhood have the same 2D velocity, more gradient constraints of the nearby pixels can be added to constitute an over-determined system.

The Lucas-Kanade method [LK81] obtained a compromise solution by minimizing the sum of square errors

$$E(\mathbf{u}) = \sum_{\mathbf{X}_i \in \mathbf{S}} [\nabla I(\mathbf{x}_i, t) \cdot \mathbf{u} + I_t(\mathbf{x}_i, t))]^2 \tag{2.4}$$

where S denotes the set of all the pixels of a neighborhood centered around the target pixel. By setting the first derivatives of the sum of squared errors to zero, we have

$$\begin{aligned} \frac{\partial E(\mathbf{u})}{\partial u_x} &= \sum_{\mathbf{X}_i \in S} [u_x I_x^2 + u_y I_x I_y + I_x I_t] = 0 \\ \frac{\partial E(\mathbf{u})}{\partial u_y} &= \sum_{\mathbf{X}_i \in S} [u_y I_y^2 + u_x I_x I_y + I_y I_t] = 0 \end{aligned} \tag{2.5}$$

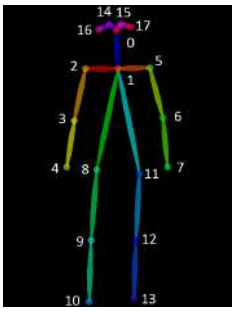
The least square estimate of optical flow field is

$$\hat{\mathbf{u}} = \begin{bmatrix} \sum\limits_{\mathbf{x}_i \in S} I_x^2 & \sum\limits_{\mathbf{x}_i \in S} I_x I_y \\ \sum\limits_{\mathbf{x}_i \in S} I_x I_y & \sum\limits_{\mathbf{x}_i \in S} I_y^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} -\sum\limits_{\mathbf{x}_i \in S} I_x I_t \\ -\sum\limits_{\mathbf{x}_i \in S} I_y I_t \end{bmatrix} \tag{2.6}$$

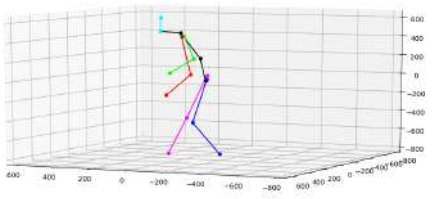
One of the most commonly used algorithms of optical flow computation is proposed by Brox et al in [BM11]. The author imposed penalizations on intensity error, gradient error, smoothness of optical flow field, descriptor matching task and descriptor matching error, resulting in a very sophisticated optimization problem.

2.1.3 Pose

Human pose, also known as human skeleton, is a high-level feature which represents the human body in a articulated system of multiple body joints. Pose can be either 2D (image coordinates) like shown in Figure 2.4(a) or 3D (with depth information) shown in Figure 2.4(b) and can be hard coded where the joint positions of the human in the video are annotated manually. One popular method of annotation is the puppet method [JGZ+13], which proposed a puppet to fit to human posture and recorded the corresponding joint locations of the puppet. While a convolutional pose machine is introduced by [WRKS16] to directly estimate human pose from videos. As is shown in Figure 2.5, the joint positions provide a meaningful representation of the human body with posture information.



(a) 2D pose skeleton[1]



(b) 3D pose skeleton[2]

Figure 2.4: Example of 2D and 3D pose

[1] https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/output.md
[2] https://github.com/SrikanthVelpuri/tf-pose

Figure 2.5: Pose cue

Compared to RGB and optical flows, pose is robust against variation of appearance and background clutters. In addition, 3D pose inherits the quality of viewpoint invariance, which facilitates the learning of cross-view features among pose representations in different viewpoints. Nowadays with the prevalence of depth camera and increasing sophistication of pose estimation algorithms, pose data are much easier to obtain, which paves the way for machine-learning-based, especially deep-learning-based algorithms for human pose estimation.

2.2 Robot Operating System (ROS)

ROS is a Linux-based, open-source, middleware framework for modular use in robot applications. ROS, originally designed by Willow Garage and currently maintained by the Open Source Robotics Foundation, is a powerful tool because it utilizes object-oriented programming, a method of programming organized around data rather than procedures in its interaction with data and communication within a modular system [Dat18]. ROS is divided into three conceptual levels: the filesystem level, the computation graph level and the community level.

2.2.1 Filesystem Level

The filesystem level is the organization of the ROS framework on a machine, as shown in Figure 2.6.

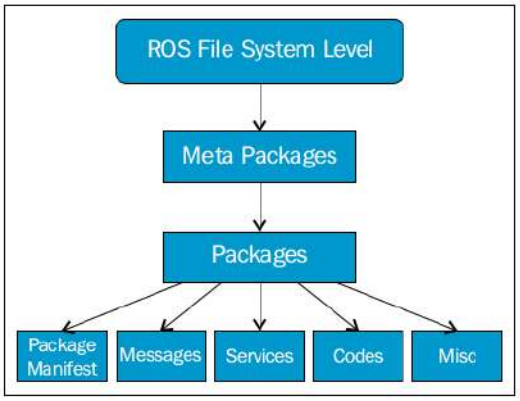


Figure 2.6: ROS filesystem level

Here are the explanations of each block in the file system:

Package : The ROS packages are the most basic unit of the ROS software. It contains the ROS runtime process (nodes), libraries, configuration files, and so on, which are organized together as a single unit. Packages are the atomic build item and release item in the ROS software.

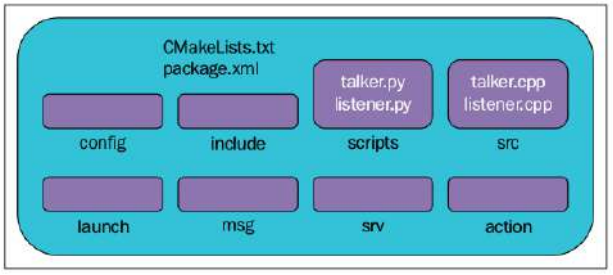


Figure 2.7: Structure of a typical ROS package

Figure 2.7 shows a typical structure of a ROS package.

We can discuss the use of each folder as follows:

- `action`[3]: Folder containing Action types, which define the Goal, Feedback and Result of the action
- `config`: All configuration files that are used in this ROS package are kept in this folder. This folder is created by the user and is a common practice to name the folder config to keep the configuration files in it.
- `include/package_name`: C++ include headers
- `launch`: This folder keeps the launch files that are used to launch one or more ROS nodes.
- `msg`: Folder containing Message types, which define the structure of data for messages sent via ROS
- `src`: Source files, including Python source that are exported to other packages and C++ source file
- `srv`: Folder containing Service types, which define the request and response data structures for services
- `scripts`: executable scripts, mostly python nodes
- `CMakeLists.txt`: CMake build file
- `package.xml`: Package manifests, providing package metadata, such as the name, author, version, description, license, dependencies, compilation flags and so on.

Message (`.msg`): The ROS messages are a type of information that is sent among ROS nodes. We can define a custom message inside the msg folder inside a package. The extension of the message file is `.msg`. The message definition can consist of two types: `fields` and `constants` . The field is split into field types and field name. Field types is the data type [4] of the transmitting message and field name is the name of it. The constants define a constant value in the message file. An example of a `msg` file is as follows, shown in Tabel 2.1:

Services (`.srv`): The ROS service is a type of request/response communication between ROS nodes. One node will send a request and wait until it gets a response from the other. The request/response communication is also using the ROS message description, which can be defined inside the srv folder inside the package. An example service description format is as follows in Tabel 2.2:

[3] `http://wiki.ros.org/actionlib`
[4] `http://wiki.ros.org/msg`

```
int32      id
float32    vel
string     name="car"
```

Table 2.1: An example of message definition

```
#Request message type
string    req_str
---
#Response message type
string    res_str
```

Table 2.2: An example of service definition

The first section is the message type of request that is separated by `---` and in the next section is the message type of response. In these examples, both Request and Response are strings.

Meta packages : Meta packages are specialized packages in ROS that only contain one file, that is, a package.xml file. It doesn't contain folders and files similar to a normal package. One of the examples of a meta package is the ROS navigation stack [5]. Meta packages simply group a set of multiple packages as a single logical package.

Repositories : Most of the ROS packages are maintained using a Version Control System (VCS) such as Git [6] and SVN [7]. The collection of packages that share a common VCS can be called repositories. The package in the repositories can be released using a catkin [8] release automation tool called `bloom`[9].

2.2.2 Computation Graph Level

The computation graph level is where ROS processes data within a peer-to-peer network. The basic elements of ROS's computation graph level are ROS **Nodes**, **Master**, **Parameter Server**, **Messages**, **Topics**, **Services** and **Bags**[10]. Figure 2.8 provides an overview of the ROS computation graph.

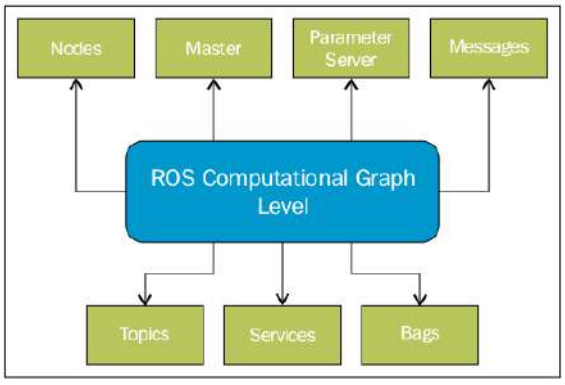


Figure 2.8: Structure of the ROS Graph layer

[5] `http://wiki.ros.org/navigation`
[6] `https://github.com/`
[7] `https://subversion.apache.org/`
[8] `https://catkin-tools.readthedocs.io/en/latest/`
[9] `http://wiki.ros.org/bloom`
[10] `http://wiki.ros.org/rosbag`

Master The ROS Master provides naming and registration services as well as `Parameter Server` to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other, they communicate with each other peer-to-peer [Wu18]. In a distributed system, the ROS master should be running on one computer, and remote nodes launched on other machines should register to that master via an environment variable `ROS_MASTER_URL`[11], which contains the IP and port of ROS Master, only then can the communication between nodes take place.

Nodes A node is a process that performs computation and serves as the atomic unit in the computation graph. Since ROS is designed to be modular at a fine-grained scale, an application or pipeline usually comprises multiple nodes. For instance, in a robot control system, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system and so on. One ROS package could also hold multiple nodes, which are written either in C++ (using roscpp[12]) or Python (using rospy[13]). Moreover, one single node can provide `Publishers`, `Subscribers` as well as `Services`, which will be discussed in the following.

- `Publisher and Subscriber`
  ROS nodes communicate with each other by publishing messages to topics, which will be subscribed by other nodes. As is shown in Figure 2.9, the first step, Node 1 (publisher) and Node 2 (subscriber) commit registration to ROS Master. The publisher set in Node 1 publishes messages of certain message type to some topic, then other nodes, which have subscribed to the same topic, can receive the messages at a pre-defined rate. Based on this publish/subscribe mechanism, ROS nodes share a high degree of flexibility and can be easily replaced, as long as the alternative nodes publish messages of the same type to the same topic or subscribe to the same topic and process the same type of message after subscription. One example in computer vision is, the publishers send image messages (`sensor_msgs/Image`) to topic `image_topic/image_raw`, and the subscribers take the image messages and process it for different purposes, while the images can be either captured online or read from local files.

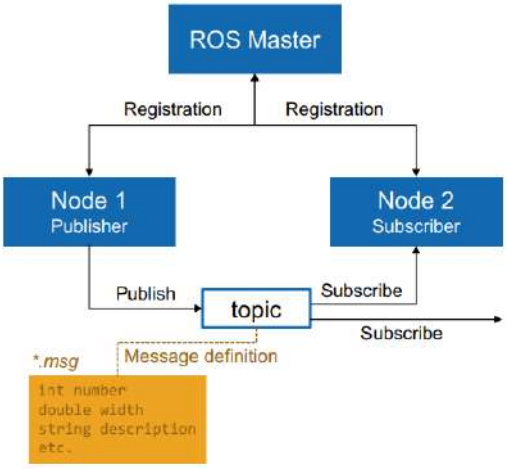


Figure 2.9: ROS publisher and subscriber communication [PF18]

[11] `http://wiki.ros.org/ROS/Tutorials/MultipleMachines`
[12] `http://wiki.ros.org/roscpp`
[13] `http://wiki.ros.org/rospy`

- `Nodelet`
  Nodelets are designed to provide a way to run multiple algorithms on a single machine, in a single process, without incurring copy costs when passing messages intraprocess. roscpp has optimizations to do zero copy pointer passing between publish and subscribe calls within the same node. To do this nodelets allow dynamic loading of classes into the same node, however they provide simple separate namespaces such that the nodelet acts like a seperate node, despite being in the same process [IS]. ROS nodelet is useful when multiple processes need to use messages which contain large amounts of data (e.g. images or point clouds), as packaging the message, sending it and then unpackaging it can take a bit of time. Nodelet makes the resource (data) shared among nodes in the same process (`NodeletManager`[14]). Nodelets don't make the processes quicker, but it is a quicker way to get information (messages) from one node to another node, avoiding copying and network traffic.

- `Service`
  The publish/subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for RPC request/reply interactions, which are often required in a distributed system. Request/reply is done via a service, which is defined by a pair of messages: one for the request and one for the reply. A server ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply [Kou18]. Figure 2.10 presents how ROS service functions.

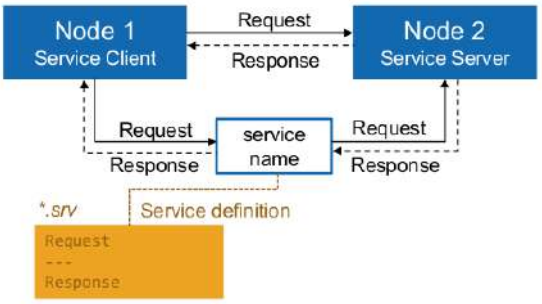


Figure 2.10: ROS service [PF18]

Parameter Server ROS parameter server is a shared, multi-variate dictionary that is accessible via network APIs. As a part of ROS Master, parameter server allows the values of parameters to be stored and retrieved by nodes at runtime [Tho13]. This is useful to configure the nodes before launched. The parameters can be defined in `launch files` or seprated `YAML files` in `config` folder within the package.

- `Launch file`
  Many ROS packages come with "launch files", which can be run with:

  ```
  $ roslaunch package_name file.launch
  ```

  These launch files usually bring up a set of nodes for the package that provide some aggregate functionality. As mentioned above, in launch file, different values for different parameters can be configured. An example is presented below, which launches three nodes: `cv_camera_node`, `person_detector`, `person_tracker` and sets three parameters in `cv_camera_node` using `<param name="" value=""/>` paradigm.

[14] `http://wiki.ros.org/nodelet/Tutorials/Running%20a%20nodelet`

```
<launch>
    <arg name="video_file" value="demo.mp4"/>
    <!-- video stream input -->
    <node pkg="cv_camera" name="cv_camera" type="cv_camera_node" output="screen">
        <param name="file" value="$(arg video_file)"/>
        <param name="rate" value="0.5"/>
        <param name="wait_for_subscriber" value="true"/>
    </node>
    <!-- person detection node -->
    <node pkg="person_detector" name="person_detector" type="person_detector"
output="screen"/>
    <!-- person tracking node -->
    <node pkg="person_tracker" name="person_tracker" type="person_tracker"
output="screen"/>
</launch>
```

- `YAML file`
  Another way to configure node parameters is to use `YAML` file as follows:
  `config.yaml`
  ```
  ------------
  file:  demo.mp4
  rate:  0.5
  wait_for_subscriber:  true
  ```

Rqt User Interface Rqt is a software framework of ROS that implements the various GUI tools in the form of plugins. Here we would like to mention two of them: `rqt_graph` and `rqt_console`.

- `rqt_graph`
  `rqt_graph` provides a GUI plugin for visualizing the ROS computation graph, including all relevant nodes and topics. Figure 2.11 shows an example of `rqt_graph`: node `cv_camera` publishes image messages to topic `/cv_camera/image_raw`, and `image_view` node subscribes to the same topic.

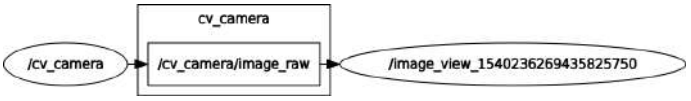


Figure 2.11: An example of rqt_graph

- `rqt_console`
  Rather than computation graph, `rqt_console` provides a GUI plugin for displaying and filtering ROS messages, like displayed in Figure 2.12.

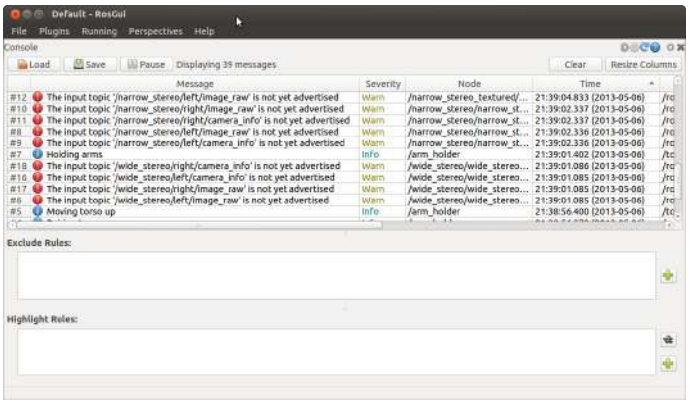


Figure 2.12: An example of rqt_consoles

`rqt_graph` and `rqt_console` are very useful tools for development, testing as well as debugging nodes, therefore we use it a lot during the development of the pipeline and later we will take advantage of them to showcase the pipeline architecture and its APIs.

2.2.3 Community Level

The ROS Community Level consists of ROS distributions, repositories, ROS Wiki, and ROS Answers, which encourages the communication and resource-sharing among robotics community across the world [Rom14].

3. Related Work

This chapter reviews some state-of-the-art works in the field of human detection and tracking, human action recognition as well as temporal action detection.

3.1 Person Detection

The goal of object detection is to detect and localize a class of object in image or video. Person detection, as a specialized case of object detection, is a crucial pre-processing step for many applications in computer vision, e.g. visual surveillance, autonomous vehicles and automated video analysis, which is exactly the first step in our pipeline (Section 4.2.3).

Early works [MPP01, VJS03] detect pedestrians using Haar-like features which have been successfully applied in face detection [VJ01]. The Haar-like features are rectangular features similar to Haar basis functions which have been used by Papageorgiou et al. [POP98]; These simple features can be computed very rapidly using an the integral image. Viola et al. in [VJS03] trained a human detector using an Adaboost algorithm, which selects a small number of classifiers iteratively from a large set of possible weak classifiers. The selected weak classifiers are combined into an effective classifier for detection. Mohan et al. [MPP01] present a detection framework that uses four example-based detectors to localize the head, legs, and arms on human body. Although these methods are very efficient, the Haar-like features are not capable of handling the complexity in real-world images and videos.

Background subtraction [EHD00, SB03, SG99, TKBM99] is also an efficient way to detect moving objects. It extracts the foreground regions by calculating the difference between a video frame and the background model. However, it has a few limitations: it requires the video is taken by a static camera - only moving objects will be detected; it is difficult to separate humans in a group; and since it does not have a specific appearance model for the target, it cannot differentiate humans from other moving objects, for example cars or animals.

In 2005, Dalal and Triggs [DT05] proposed a Histogram of Oriental Gradients descriptor (HOG) for object detection. In their method, a set of overlapping HOG features are extracted from a sliding window and then fed into an SVM [CV95]. The advantage of HOG features is that each image cell is statistically represented by a histogram of the gradient orientations and magnitudes, thus it is more invariant to illumination, shadows, etc. Wang and Han [WHY09] proposed a HOG based detection framework that combines

HOG features and Local Binary Pattern (LBP) [AHP06] feature, which captures the subtle intensity changes in a small local area. Due to its superior performance and computational efficiency, LBP has become a popular feature for texture classification, face recognition, and object detection. Based on video, additional information such as optical flow or motion constraints can greatly assist the detection task. Dalal et al. [DTS06] proposed a new histogram of optical flow feature and combined it with HOG feature. However, the computational cost of calculating motion features is relatively high.

With the success of deep learning methods in recent years [KSEH12], detectors based on CNNs have been shown to outperform those traditional approaches. A seminal work in this area is R-CNN proposed by Girshick et al. [GDDM13], which makes use of a classification network by applying it to a set of pre-computed object proposals and feeding the extracted features to class-specific SVMs in order to classify each region independently.

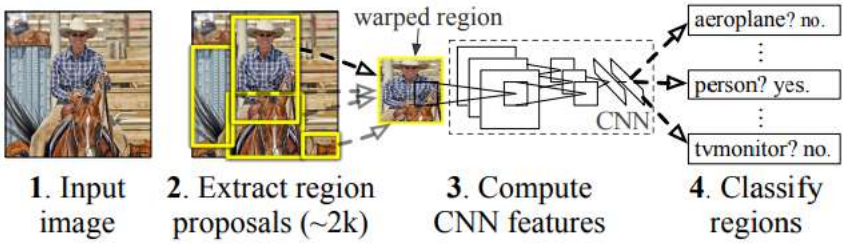


Figure 3.1: **R-CNN object detection system overview.** (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class- specific linear SVMs. [GDDM13]

Its successors Fast R-CNN [Gir15] and Faster R-CNN [RHGS15] further improve over it by sharing computation on single images with region of interest (RoI) pooling and integrating the region proposal generation into the network, respectively.

Another two popular CNN based detectors are here to mention: Single Shot Multibox Detector (SSD) [LAE+15] and You Only Look Once (YOLO) [RDGF15], which operate in a fully-convolutional manner and do not rely on region proposals to generate detection hypotheses. The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections [LAE+15]. Figure 3.2 shows how the SSD framework detects objects in the scene and Figure 3.3 presents how YOLO model works.

3.2 Person Tracking

Person tracking is the task of tracking a person over several frames of an image sequence. It can be divided into two categories: Single Object Tracking (SOT) and Multiple Object Tracking (MOT).

3.2.1 Single Object Tracking

Early works [BC86, ST94] focus on estimate the motion parameters of the target. Shi et al. [ST94] uses the affine motion model to compensate the image motion and selects discriminative feature points for the tracker. Comaniciu et al. [CRM03] proposed the mean-shift algorithm to track objects by finding the peak in a confidence map of the surrounding area. This confidence map is a probability density function calculated on each

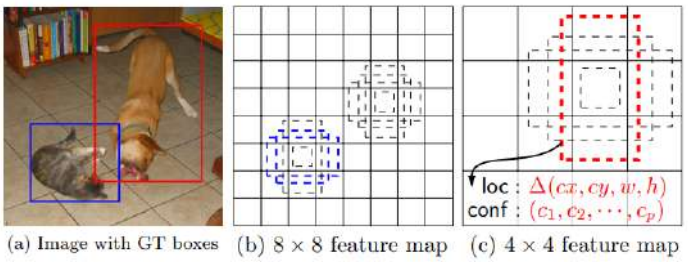


Figure 3.2: **SSD framework.** (a) SSD only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, a small set (e.g. 4) of default boxes of different aspect ratios at each location in several feature maps with different scales is to be evaluated (e.g. 8×8 and 4×4 in (b) and (c)). For each default box, both the shape offsets and the confidences for all object categories ($(c_1, c_2, \cdots, c_p)$) are predicted. [LAE+15]

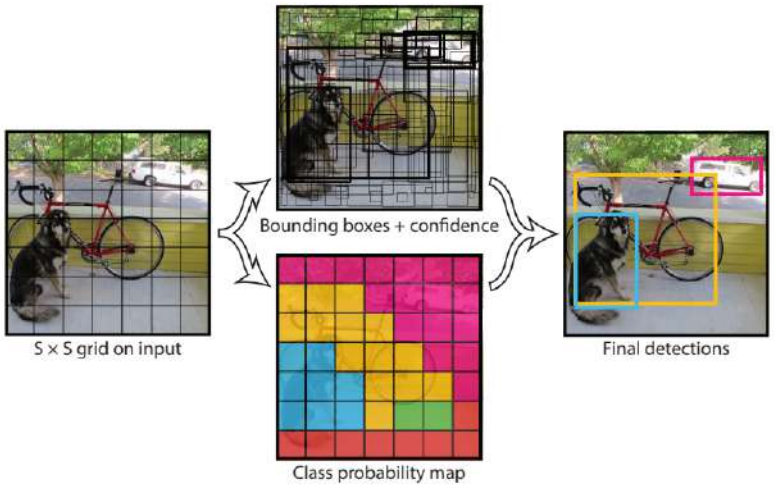


Figure 3.3: **The YOLO model.** The system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B boundingboxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensors. [RDGF15]

pixel using color similarity. The above-mentioned works provide basic tracking frameworks. However, the object models in these methods are too simple to handle real world videos.

Later discriminative tracking approaches with online-learning have been extensively explored. Avidan [Avi07] train an ensemble of weak classifiers using AdaBoost to distinguish between the object and the background. Collins and Liu [CLL05] selects the best features to track based on the two-class variance ratio between foreground and background. Grabner et al. [GGB06] learns a discriminative classifier in an online manner to detect the object from the background. Babenko et al. [BYB09] use Multiple Instance Learning (MIL) to avoid the drift caused by inaccurate bounding boxes.

In such methods, an object-specific detector is trained in a semi-supervised fashion and then used to locate the target in consecutive frames. In particular, a few positive and negative examples are collected each time the target is located, and these examples are immediately used to re-train the detector. In this way, the detector can capture the most discriminative features on the object. However, the online learned detector will often drift into the background in long-term tracking.

In recent years, a few sophisticated methods for single-target tracking have been proposed to address complex real world data. Kala [KMM10] leverages the video structure to select the most confusing negative examples from backgrounds to strengthen the classifier. Dinh et al. [DVM11] tracks multiple objects beside the target, therefore it avoids the confusions with other objects with similar appearances. Hare et al. [HGS+16] present a framework for adaptive visual object tracking based on structured support vector machine, which provides superior tracking performance. Another well-known tracking method raised by R.E.Kalman in 1960 is the Kalman filter [WB95], which is mostly used to solve the discrete-data linear filtering problem and has been applied in various research works like [VVV15] and [KST+06].

With sophisticated models and online-learning approaches, these methods are more invariant to appearance change and occlusion. However, their computational cost is relatively high for a practical tracking system. Moreover, in multi-person scenarios, SOT is not able to track multiple agents in real world video surveillance.

3.2.2 Multiple Object Tracking

A large number of works have focused on multi-target tracking-by-detection algorithms. These methods tackle multi-target tracking by optimizing detection assignments over a temporal window, given certain global constraints. Blackman [Bla04] proposed a multiple hypothesis tracking framework: possible trajectory hypotheses are proposed and propagated into the future in anticipation that subsequent data will solve the difficult data association decisions. Zhang et al. [ZHZ08] resolve the association between the detection and the tracking by optimizing a cost-flow network with a non-overlap constraint on trajectories. Brendel et al. [BAT11] apply a maximum-weight independent set algorithm to merge small tracklets into long tracks. Benfold et al. [BR11] use Markov-Chain Monte-Carlo Data Association (MCMCDA) to correspond the detections obtained by a HOG-based head detector in crowded scenes in multi-threads. Such methods employ offline-trained detectors to find the targets and associate them with the tracks. Another approach is developed by C.Kuo [KHN10] for online learning of discriminative appearance models for robust multi-target tracking in a crowded scene. Within a time sliding window from tracklets, training samples are assembled online and also for combining effective descriptors and their similarity measurements, the AdaBoost algorithm is used.

3.3 Action Recognition

According the input modalities used for human action recognition, related works can be categorized into non-pose-based (Section 3.3.1) and pose-based (Section 3.3.2) approaches. The non-pose-based action recognition, which have been intensively studied over the past decades, employ features extracted from RGB image, depth map or optical flows. Pose-based action recognition, which is also the focus of this thesis, takes advantage of the high-level skeleton information.

3.3.1 Non-Pose-based Action Recognition

For solving human action recognition problem, researchers presented some approaches that are based on RGB frames and optical flows. They implemented the techniques to capture context information from RGB frames and motion information (e.g. optical flow). Some researchers presented their work in domain of conventional machine learning approach, i.e, extraction of handcrafted features from videos, encoding them with bag-of-words [ZJZ10] and then training a classifier, such as SVM [CV95], with these extracted descriptors to perform action classification. Apart from conventional machine learning approaches, many research works have also utilized deep learning for action classification. In the following, both handcrafted-feature-based methods and deep-learning-based methods will be discussed.

3.3.1.1 Handcrafted-Feature-based Approaches

Before deep learning came out to dominate in the field of computer vision, the traditional pattern recognition problems rely on hand-crafted feature extractors, so does the task of action recognition. Most of approaches usually follow the classical pipeline of pattern recognition to first extract low-level appearance features like image gradients, or motion features such as optical flows and feature descriptors such as HOG [DT05], HOF [LMSR08] or SIFT [Low04], SURF [BETG08]. Then mid-level features are extracted through sparse dictionary learning or bag of words. Finally, the features are trained with a classifier or regressor in a supervised manner.

Improved Dense Trajectories

The work of improved dense trajectories [WS13] is acknowledged as the handcrafted feature-based approach owing to its great performance on several action recognition benchmarks.

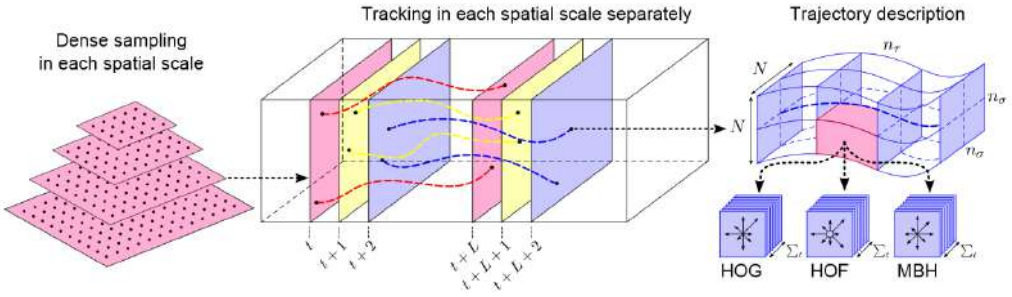


Figure 3.4: Left: Dense sampling of feature points in different spatial scales. Middle: Tracking in an optical flow field. Right: descriptors along the trajectory. [WS13]

As is shown in Figure 3.4, feature points at different spatial scales are densely sampled in the image. Points in homogeneous areas where no visual structure is available are removed since it is impossible to track any point on these areas. The second step is the tracking of the feature points along the temporal dimension in the dense optical flow field, in which there is a displacement vector at every single position that implies where the point has moved in the next frame. Along each trajectory, HOG, HOF and MBH (motion boundary histograms) [DTS06] descriptors are extracted in the local coordinate system centered around the keypoint. At last, the action classification is performed in a standard pattern recognition pipeline: pooling of bag-of-features representation (generation of codebooks in an unsupervised manner) and then training the descriptors with an SVM classifier.

This work is an improvement upon [WKSL11] from the same author. Based on the former work, it compensates camera motion by matching feature points between frames using SURF descriptors and dense optical flow and estimating a homography with RANSAC [FB81]. On the challenging action datasets UCF50 [RS13] and HMDB51 [KJG+11], this approach of improved dense trajectory [WS13] has achieved the state-of-the-art performance among all the non-deep-learning-based methods.

3.3.1.2 Deep-Learning-based Approaches

With the advanced performance of convolutional neural networks (CNN) in image classification [SZ14b], researchers utilized CNN for video recognition tasks. Unlike single static image classification, both the spatial and temporal pattern should be encoded inside the 3D volume in the scenario of video analysis.

Two Stream Convolutional Neural Networks

Simonyan et al. [SZ14a] proposed a two-stream ConvNet architecture, as is illustrated in Figure 3.5, which models the temporal information locally is to exploit handcrafted features that contain local motion information. The spatial stream operates on individual RGB frames while the temporal stream operates on stacking of optical flows of multiple frames. The softmax scores of both streams are fused lately at the very end of the architecture and it turns out that the fusion of two streams has better performance than either stream alone, which implies that temporal and spatial streams are complementary. Based on the same two-stream architecture, [FPZ16] studies different approaches of fusing spatial and temporal CNNs. According to the experiment results, it is better to fuse the two networks at the last convolutional layer than earlier. It also proposes an additional fusion in the class prediction layer to further boost the accuracy.

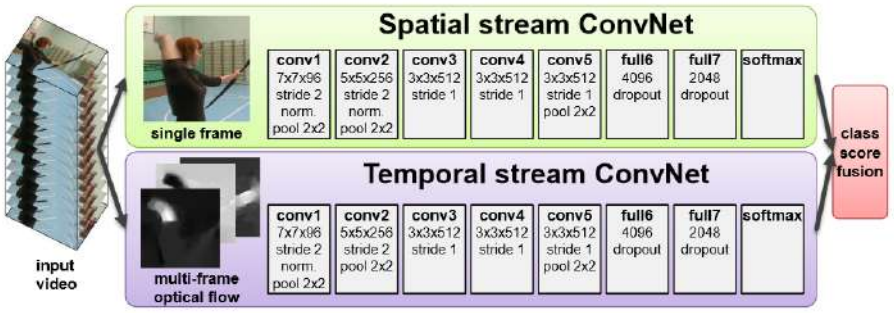


Figure 3.5: Two-stream structure for video classification. [SZ14a]

Temporal Segment Networks

Another successful application of adapted two-stream network was the Temporal Segment Networks (TSN) [WXW+16a]. TSN is also built upon the conventional two-stream architecture as shown in Figure 3.6.

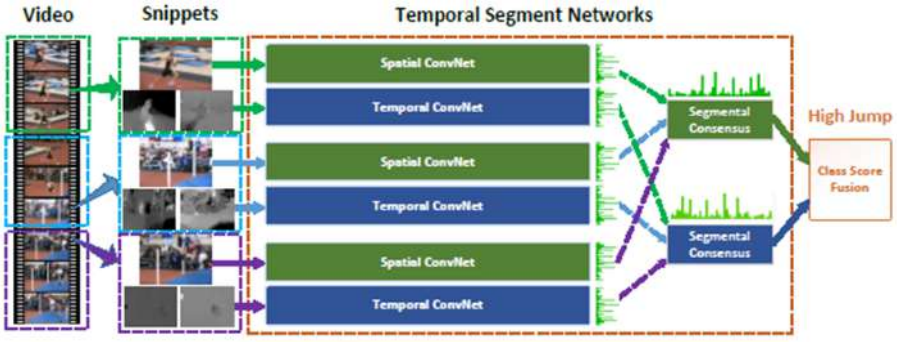


Figure 3.6: Complete pipeline of Temporal Segment Networks. [WXW+16a]

However, unlike [SZ14b], it has proposed a sparse and global temporal sampling strategy to extract short snippets over the entire video sequence. Each video sequence is divided into several equal-lengthed segments and a short snippet is randomly selected from each segment to remove redundant information in consecutive frames. Each snippet is represented by several modalities such as RGB images, stacked optical flow fields, stacked RGB differences and stacked warped optical flow fields. Then the class scores of all the snippets are aggregated with an segmental consensus function and softmax predictions of

all the modalities are fused at the end to determine the final class scores. The segmental consensus is derived from all the snippet-level prediction. By back propagation with the loss function upon segmental consensus, the network is optimized using visual information from the entire video. TSN has also achieved the state-of-art performance on two challenging action recognition benchmarks HMDB51 [KJG+11] and UCF101 [SZS12].

Apart from TSN, recurrent neural network (RNN) is a more conventional way to encode long-term temporal. Donahue et al. [DHG+14] has leveraged CNNs for extraction of visual features and Long Short-Term Memory (LSTM) [HS97] for sequential learning of the features, leading to predictions of every single frame of a video. Similarly, the work from Google research [NHV+15] uses CNNs for extraction of features from RGB images and optical flows. Then it employs an LSTM for feature aggregation.

3.3.2 Pose-based Action Recognition

A human body can be represented as an articulated system of rigid limbs connected by several body joints and human action can be regarded as a continuous temporal evolution of the spatial configuration of these body joints. A distinct gesture together with certain movement of 2D/3D body joints can highly predicts the intention of the human subject. Under this precondition, the main idea of pose-based action recognition is to track the movements and displacements of human body joints in every frame and then classify the temporal evolution of the human skeletons.

2D Human Pose Estimation by Part Affinity Fields

Cao et al. [CSWS17] proposed an efficient technique to detect 2D pose of multiple persons in the image. The technique comprises of two branch multi-stage CNNs. The architecture of this technique is shown in Figure 3.7. The branch shown in beige colour is trained to predict the confidence maps S of joint locations. The second branch shown in blue colour, is trained to predict the affinity fields L between the adjacent joints. The affinity fields is a 2D vector field, which points from one joint location to adjacent joint, as shown in Figure 3.8(c). Each branch is trained with intermediate supervision at each stage, that ensures the refinement of predictions over successive stages.

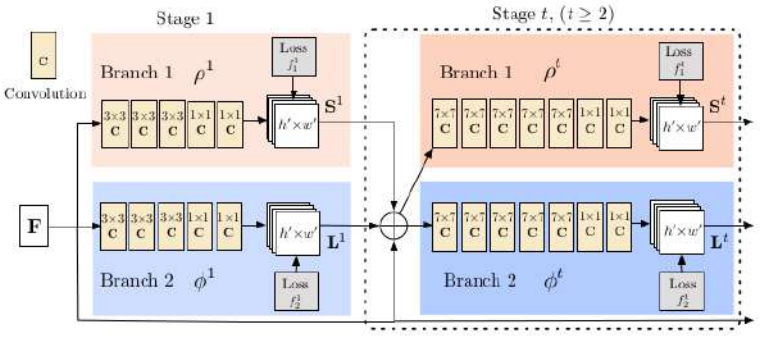


Figure 3.7: Multi stage two branch architecture for pose estimation by part affinity field. [CSWS17]

During training, the image is first analysed by standard VGG-19 [VL15] convolutional neural network, which generates a set of feature maps F. In the first stage, these features maps F serve as inputs to both branches. These branches produces a set of confidence maps predictions S^1 and a set of affinity fields predictions L^1 in the first stage. In the

Figure 3.8: Overall pipeline of pose estimation by part affinity field technique. [CSWS17]

subsequent stages, both of these predictions S^1, L^1 along with feature maps F, are concatenated and used to produce refined predictions. The network is trained on large COCO dataset [LMB+14]. The trained network takes RGB images as inputs and predicts confidence maps of each joint along with part affinity fields as shown in Figure 3.8(b),(c). The final pose of each person is found by part association algorithm. In this algorithm, the association between candidate part detection is measured by computing the line integral over the corresponding part affinity field, along the line segment connecting the joint locations. Our pipeline utilizes the OpenPose[1], which is based on this work, as the 2D pose estimator.

Monocular 3D Pose Estimation

Another pose estimation framework is proposed in [ZHS+17], which is a weakly-supervised approach for 3D pose estimation from monocular RGB frames. The framework architecture is illustrated in Fig 3.9.

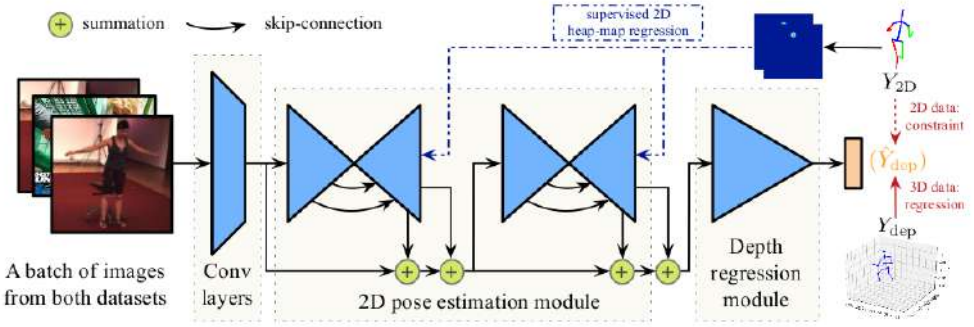


Figure 3.9: The stacked hourglass network computes the 2D heat maps (2D joints) based on the image features extracted by Conv layers and the depth regression module computes the depth value based on summation of 2D heat maps and image features extracted in the beginning. [ZHS+17]

The proposed network splits the task into estimation of 2D joints and regression of depth value based on the estimated 2D joints. The architecture consists of a 2D pose estimation module which is imported from the state-of-art hourglass architecture in [NYD16] and a depth regression module. This architecture not only benefits from the existing sophisticated 2D pose estimation algorithms but also overcomes the domain difference between datasets in indoor laboratory setting where 3D joints are obtained by motion capture system and datasets in the wild that have annotations of 2D joints. As is shown in Figure 3.10, for 3D data, 3D Euclidean loss is applied for standard regression, while weakly-supervised loss based on 2D Euclidean loss and the sum of variance of bone length ratios in several body parts (the ratio between estimated bone length and the average bone length among

[1] `https://github.com/CMU-Perceptual-Computing-Lab/openpose`

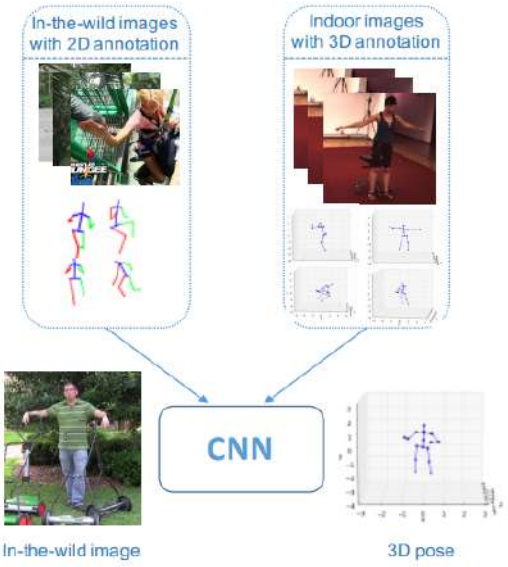


Figure 3.10: A schematic illustration of our method: transferring 3D annotation from indoor images to in-the-wild images. Top (Training): Both indoor images with 3D annotation (Right) and in-the-wild images with 2D annotation (Left) are used to train the deep neural network. Bottom (Testing): The learned network can predict the 3D pose of the human in in-the-wild images. [ZHS+17]

all training data) is applied for 2D data. As the depth information is inferred from a monocular 2D image by data-hungry learning techniques instead of triangulation of stereo vision, the estimated pose can be referred to as “pseudo-3D” skeleton data.

3.4 Temporal Action Detection

In the context of video-based human action recognition, temporal action detection and grouping is of vital importance, as videos in real applications are usually long, untrimmed and contain multiple action instances. The challenge for our pipeline lies not only in recognizing action categories based on human pose and other features, but also detecting the start time (frame) and end time (frame) of each action instance. Many state-of-the-art methods [Gir15, GDDM13, RHGS15] adopt the ”detection by classifying region proposals” framework: first do proposal, and then classify proposals. The main drawback of this framework is that proposal generation and classification procedures are separate and have to be trained seperately. Lin et al. [LZS17] proposed a novel Single Shot Action Detector (SSAD) network based on 1D temporal convolutional layers to skip the proposal generation step via directly detecting action instances in untrimmed video. As illustrated in Figure 3.11, the SSAD network take the Snippet-level Action Score features as input and predicts the actions directly, which already contains the start and end information of the snippets.

Another action grouping strategy proposed in [XZW+17] can efficiently generate candidates with accurate temporal boundaries, while in the fashion of $proposal + classification$.

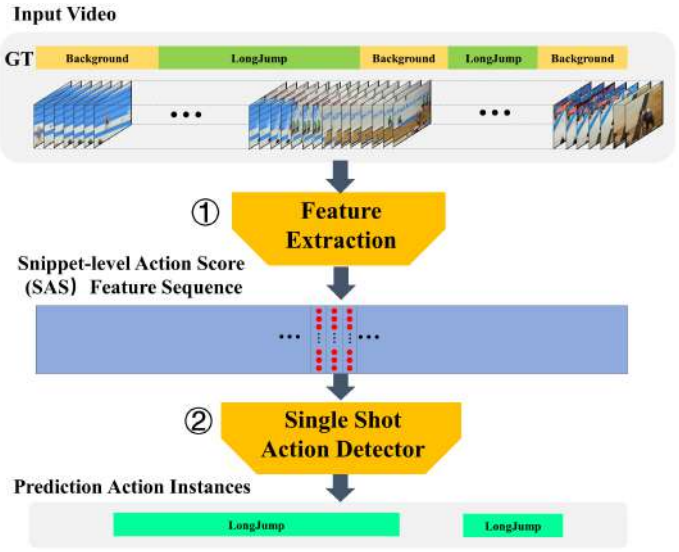


Figure 3.11: **Overview of SSAD framework**. Given an untrimmed long video, (1) Snippet-level Action Score features sequence with multiple action classifiers are extracted; (2) SSAD network takes feature sequence as input and directly predicts multiple scales action instances without proposal generation step. [LZS17]

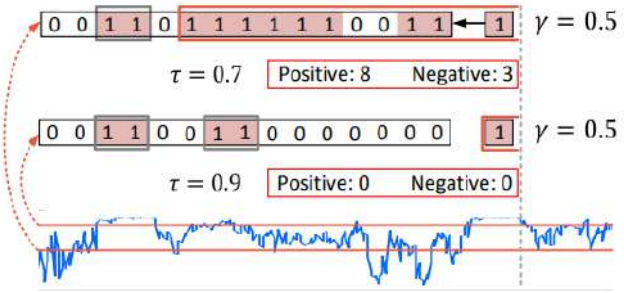


Figure 3.12: An illustration of the temporal actionness grouping algorithm. [XZW+17]

As shown in Figure 3.12, the scheme first obtains a number of action fragments by thresholding – a fragment here is a consecutive sub-sequence of snippets whose actionness scores are above a certain threshold τ. Then, to generate a region proposal, a fragment is picked as a starting point and it will be expanded recursively by absorbing succeeding fragments. The expansion terminates when the portion of low-actionness snippets goes beyond γ, a positive value which is referred to as the tolerance threshold. Beginning with different fragments, a collection of different region proposals can be obtained [XZW+17].

4. Pipeline Design

In this chapter, we will walk through the pipeline and explain how different modules interact with each other. Firstly an overview of the pipeline architecture is presented, then the functionality of different modules will be introduced. At last, from the perspective of implementation, we summarize the APIs of the pipeline for application and further development.

4.1 Pipeline Architecture

The pipeline mainly consists of eight modules, i.e. Source Provider, Person Detector, Person Tracker, Feature Extractor, Data Manager, Action Predictor, Visual Agency and Utilities, which interact with each other via ROS Message/Topic mechanism (see Section 2.2.2). We basically set up publishers and subscribers in different nodes to transport the data flow. Figure 4.1 provides an overview of the pipeine modules and the messages between them, while Figure 4.2 reveals more details about the ROS nodes and message topics between nodes, where each ellipse stands for a ROS node and the following rectangle holds the required message topics under the ROS node. Based on the modules, two scenarios will be discussed here.

Single-Person vs. Multi-Person Scenario

In the case of single agent, the RGB image from *Source Provider* could be directly fed into *Feature Extractor* – in our case, *openpose_ros* and *pose_3d_ros*, an extra person detection and tracking module is not a must. Figure 4.3 shows a trivial pipeline for single person.

Figure 4.3: A trivial pipeline for single person

However, when it comes to multi-person scenario, every detected person, as shown in Figure 4.4, should be tracked, shown in Figure 4.5, thus extracting an unique ID for each person. Then based on the boundingbox of each person after tracking, a cropped image of each person will be fed into the pose estimation model. *Data Manager* will create a pose pool to keep a track of the pose data for all the people detected and tracked, and preprocess the pose data of each person respectively. Figure 4.6 displays the final action

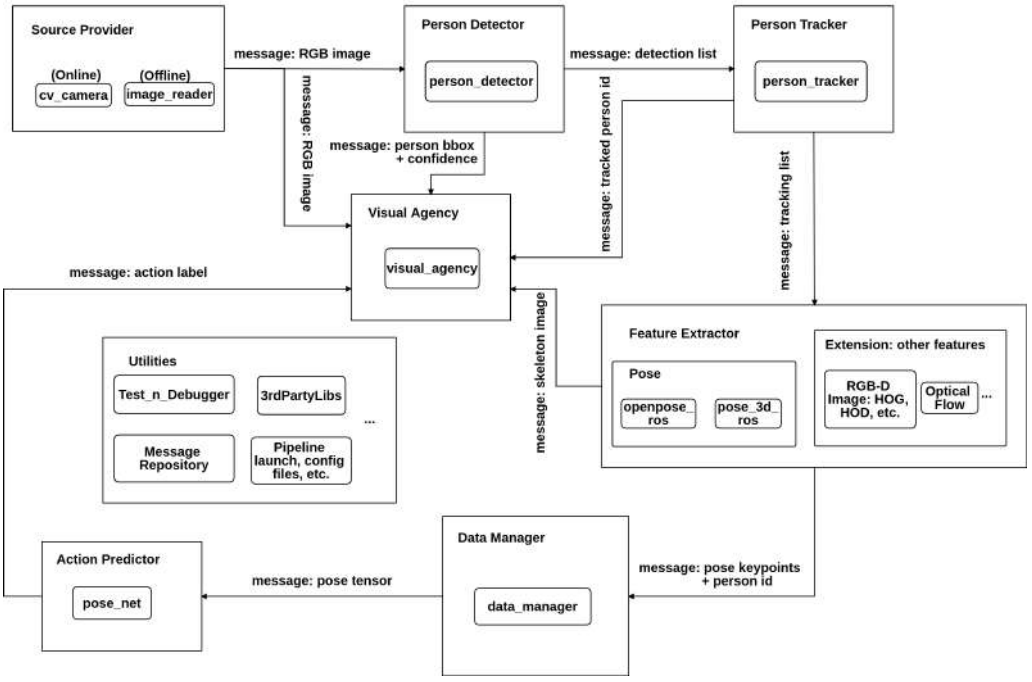


Figure 4.1: Pipeline Modules and Messages

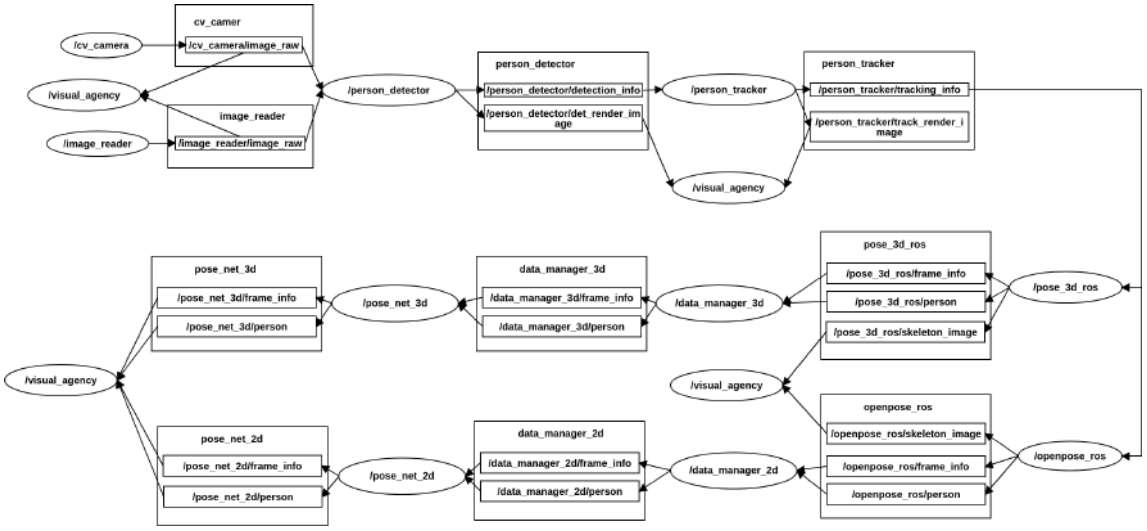


Figure 4.2: Pipeline ROS Nodes and Message Topics

classification results of each person in the scene. In our pipeline, we consider multi-person as default scenario, as a result, the trivial pipeline for single person is integrated in our final pipeline.

4.2 Module Functionality

As described above, different modules serve as different roles in the pipeline, in the following more details of each component will be discussed.

4.2.1 Message Repository

Before we go deeper into the modules, we should first introduce the messages involved in the pipeline. To manage all the ros custom messages, services as well as actions, a ros

Figure 4.4: **A detection demo scenario.** We utilize YOLO3 [RF18] as our detector and filter out all the detected persons with a confidence of less than 80%

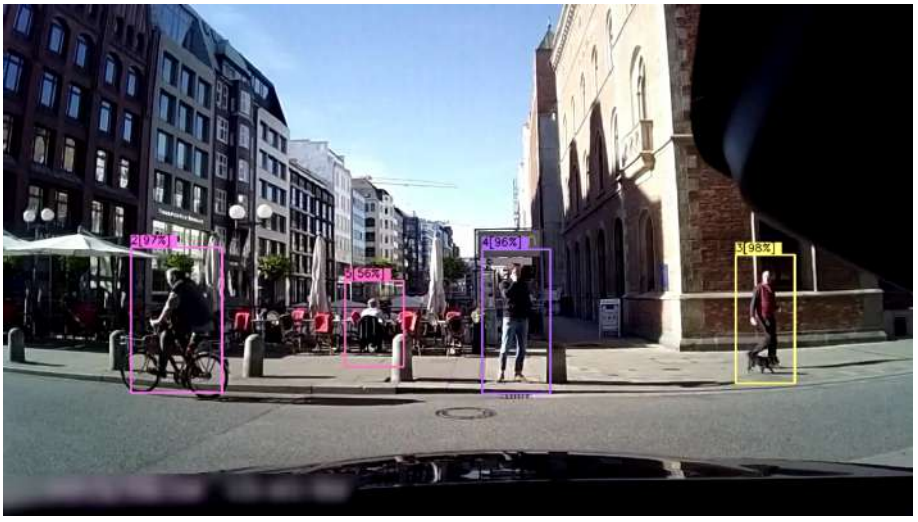


Figure 4.5: **A tracking demo scenario.** Taking the detection list as input, our tracker will track each person appeared in the scene and assign each of them an unique person ID.

package named message_repository is created. Three messages are currently used: `FrameInfo.msg`, `Person.msg` and `BoundingBox.msg`. Their definitions are listed below:

`FrameInfo.msg`

```
# Message defining frame information

uint32              frame_id     # ID of current frame
sensor_msgs/Image   image_frame  # Current RGB image frame
Person[]            persons      # All persons detected/tracked in the current frame
bool                last_frame   # signal for last frame at testing
```

The detection info and tracking info message are actually of type `FrameInfo`, which contains the the current image and its frame number, a list of detected or tracked persons in

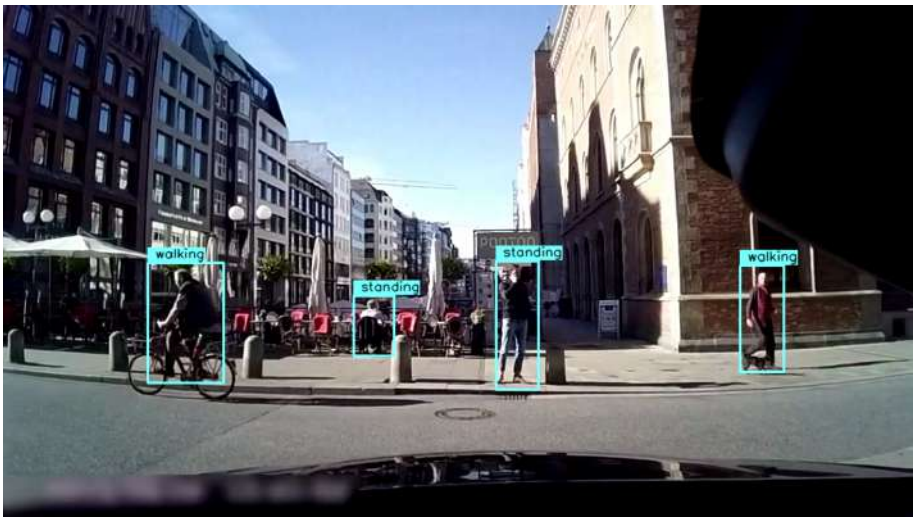


Figure 4.6: **An action demo scenario.** Our pose estimator will extract pose data for each person based on the cropped image, and then *data_manager* generates pose tensor and at last, action classifier will do the action prediction (in the example, walking or standing), taking the pose tensor as input.

the image frame and an extra `last_frame` flag for signaling the end of a video or image sequence at testing and evaluation stage.

`Person.msg`

```
# Message defining person information

uint32                    frame_id         # current frame number
uint32                    person_id        # unique identity of the person
BoundingBox               bbox             # (x_left, y_top, width, height)
float64                   confidence       # confidence of detection/tracking
geometry_msgs/Point[]     person_pose      # pose keypoints of a person in a frame
sensor_msgs/Image         pose_tensor      # pose tensor for action prediction
string                    person_action    # predicted action label
```

The `Person` message includes all the required person information along the pipeline. Actually, different modules are responsible for the different information in the `Person` message: *Source Provider* offers the input of the whole pipeline, which is an RGB image message, and it conveys the information of `frame_id`. Then *Person Detector* extracts `bounding box` of person and the corresponding `confidence`. After that, *Person Tracker* keeps a track of each person and assigns `person_id` to them. The role of *Feature Extractor* is to get the `person_pose` with the help of some deep learning models. *Data Manager* serves as data processing, which generates `pose_tensor` for specific person. At last, *Action Classifier* predicts the `person_action` based on the `pose_tensor` from *Data Manager*.

BoundingBox.msg

```
# Message defining person bounding box

float64    left      # left position of the person
float64    top       # top position of the person
float64    width     # width of the bbox
float64    height    # height of the bbox
```

The BoundingBox message plays an auxiliary role for storing the bounding box information extracted by *Person Detector*, with the pattern of (x_min, y_min, bbox_width, bbox_height), where x and y are the coordinates in the image.

4.2.2 Source Provider

As input of the whole pipeline, *Source Provider* offers two available sources, one for online video stream (using cv_camera package), the other for offline image sequence (using image_reader package).

4.2.2.1 Cv_camera Package

The cv_camera package[1] includes a ROS OpenCV camera driver, which takes advantage of cv::VideoCapture of OpenCV[2] and interacts with hardware devices (e.g. webcam, video camera) and video files. It contains two nodes: cv_camera_node and its Nodelet[3]. Figure 4.7 shows the computation graph of cv_camera node.

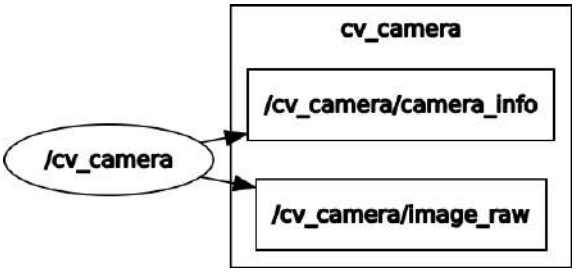


Figure 4.7: ROS Topics and Messages in cv_camera node

APIs Summary

ROS node: cv_camera_node

- Publisher
  Topic: /cv_camera/image_raw
  Message type: sensor_msgs/Image
  Message: RGB image message
- Main Parameters
  ~rate: publish rate, default: 30.0
  ~device_id: capture device id, default: 0
  ~image_width: set capture image width
  ~image_height: set capture image height
  ~file: set to use video file instead of device

[1] https://github.com/jiangyonghui/cv_camera
[2] https://docs.opencv.org/3.1.0/d8/dfe/classcv_1_1VideoCapture.html
[3] http://wiki.ros.org/cv_camera

- Main Flags
  `--wait_for_subscriber`: if true, cv_camera_node will start working until at least one subscriber is set to topic `/cv_camera/image_raw`, default: false

4.2.2.2 Image_reader Package

Similar to `cv_camera`, `image_reader` package provides a ros node, which reads image sequences from local folders or server and publish image message at a certain frequency, as shown in Figure 4.8.

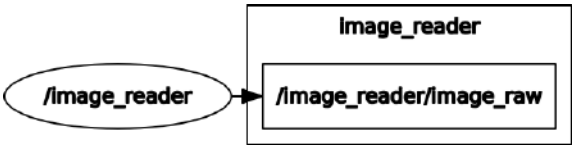


Figure 4.8: ROS Topics and Messages in `image_reader` node

APIs Summary

- Publisher
  Topic: `/image_reader/image_raw`
  Message type: `sensor_msgs/Image`
  Message: RGB image message
- Parameters
  ∼`image_path`: full path to image folder
  ∼`rate`: publish rate, default: 30.0
- Flags
  `--display_image`: set true to show the published image

4.2.3 Person Detector

The first step after input is to detect persons in the image frame. In this module, we wrap YOLO darknet[4] C++ API into the ROS node and take advantage of pretrained model[5].

4.2.3.1 YOLO Darknet

As described in Section 3.1, the YOLO network [RDGF15] divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. A detection example is shown in Figure 4.9.

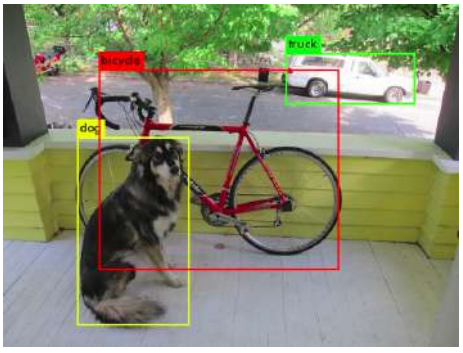


Figure 4.9: A YOLO detection example.

[4] `https://gitlab.com/EAVISE/darknet`
[5] `https://pjreddie.com/darknet/yolo/`

Based on the runtime performance as shown in Figure 4.10, we utilize the pretrained model YOLOv3-320[6], which achieves real-time in our final testing with 35 40fps (frame per second).

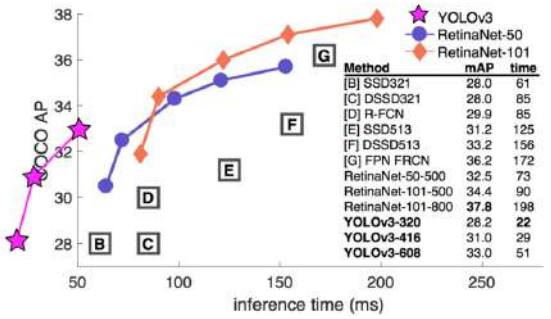


Figure 4.10: YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU. [RDGF15]

4.2.3.2 Person_detector Package

In this module we create one ROS package – `person_detector`, which includes one ros node. Figure 4.11 shows the computation graph of `person_detector` node.

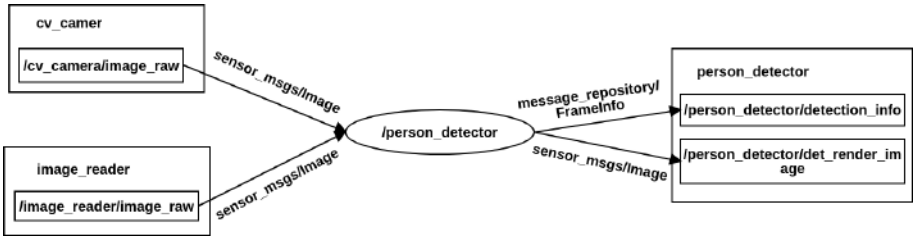


Figure 4.11: ROS Topics and Messages in `person_detector` node

Implementation Details

In the implementation, since we haven't retrained the yolo model specifically for person, an object filter is applied to only get the bounding boxes of persons, at the same time, a threshold confidence for detection is set to leave out the less potential agents in the detection list. Figure 4.12 shows an example of detecion without thresholding detection confidence, while Figure 4.4 displays the filtered detection result.

While this person detector also comes with some problems. Somehow the wrong object would be taken as person while the true human agents are left out of the detection list. As illustrated in Figure 4.13, the detected object in yellow is not a person, though it is label as one, while some other persons in the scene are not detected.

In terms of ROS, the input image message is of type `sensor_msgs/Image`, which will be converted to OpenCV image format (`cv::Mat` or `cv2.image`) via another ROS package — `cv_bridge`[7]. As shown in Figure 4.14, the `cv_bridge` package serves as the bridge

[6] `https://pjreddie.com/darknet/yolo/`
[7] `http://wiki.ros.org/cv_bridge`

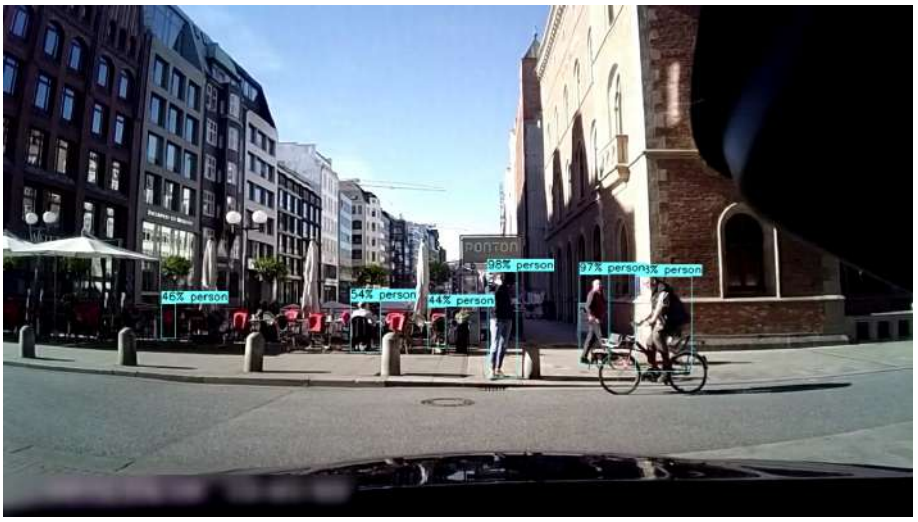


Figure 4.12: **A detection demo scenario without thresholding detection confidence.** Before thresholding the confidence, some random detection appears in the detecion list.

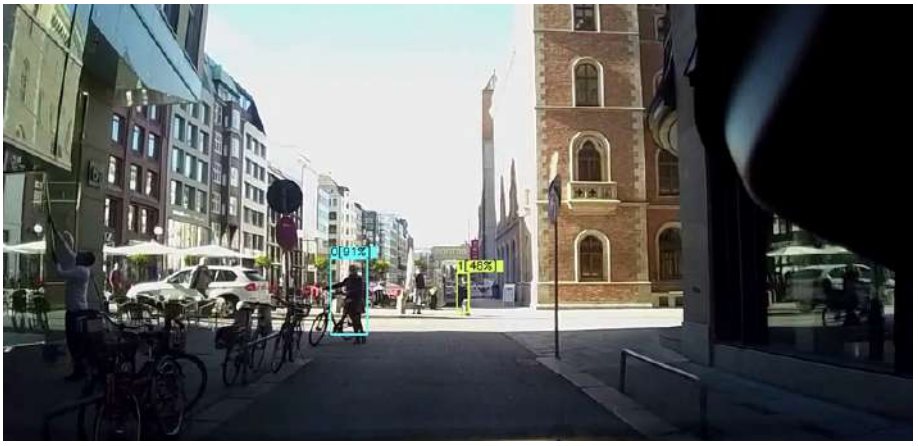


Figure 4.13: A scene of both positive false and negative false case from the YOLO person detector

between ROS image message and OpenCV image, offering the APIs for the convertion in both C++[8] and Python[9].

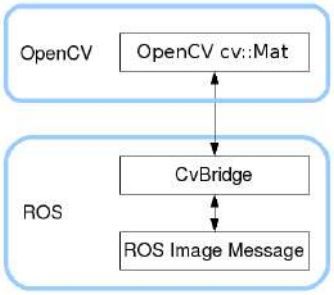


Figure 4.14: **CvBrideg**: Convertion between ROS image messages and OpenCV images. [RBR]

[8] `http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages`
[9] `http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython`

APIs Summary

- Subscriber
  Topic: `cv_camera/image_raw`; `image_reader/image_raw`
  Message type: `sensor_msgs/Image`
  Message: RGB image message
- Publisher
  Topic: `/person_detector/detection_info`
  Message type: `message_repository/FrameInfo`
  Message: detection list of persons with frame ID, image frame, person bounding box and detection confidence
  `-----`
  Topic: `/person_detector/det_render_image`
  Message type: `sensor_msgs/Image`
  Message: detection image rendered with person bounding box and confidence
- Main Parameters
  `~image_topic`: set image topic for image subscriber `~model_cfg`: network configuration, default: `"yolov3.cfg"`
  `~model_weights`: network weights, default: `"yolo3.weights"`
  `~thres_conf`: confidence threshold for filtering detected persons, default: 0.8
- Main Flags
  `--visual_detection`: set ture to display images rendered with person bounding box
  `--write_video`: set true to write detection video
  `--write_det_file`: set true to write detection list to file

4.2.4 Person Tracker

Based on the detection list from *Person Detector*, *Person Tracker* is supposed to extract the IDs for each person that has appeared in the scene and keep a track of them.

4.2.4.1 Person Tracker Framework

Our tracker utilizes the basic framework similar to [ARS06]. As described, let (x_0, y_0) be the person position estimation from the previous frame (for the first frame, it is the position from detection result), and let r be our search radius. Let $P_T = (dx, dy, h, w)$ be an arbitrarily chosen rectangular patch in the template, whose center is displaced (dx, dy) from the template center, and whose half width and height are w and h respectively. Let (x, y) be a hypothesis on the person's position in the current frame. Then the patch P_T defines a corresponding rectangular patch in the image $P_{I;}(x, y)$ whose center is at $(x + dx, y + dy)$ and whose half width and height are w and h. Figure 4.15 describes this correspondence.

Given the patch P_T and the corresponding image patch $P_{I;}(x, y)$, the similarity between the patches is an indication of the validity of the hypothesis that the object is indeed located at (x, y). If $d(Q, P)$ is some measure of similarity between patch Q and patch P , then we define

$$V_{P_T}(x, y) = d(P_{I;}(x, y), P_T) \tag{4.1}$$

When (x, y) runs on the range of hypotheses, we get $V_{P_T}(\Delta, \Delta)$, which is the vote map corresponding to the template patch P_T. We measure similarity between patches by comparing various features (e.g. FHOG [FGMR10]) extracted from them. After that, the vote

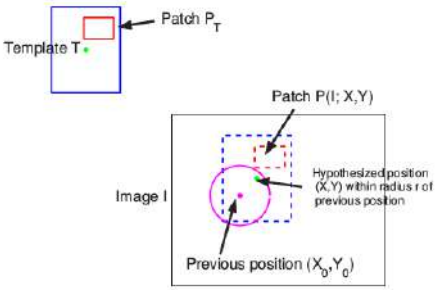


Figure 4.15: Template patch P_T and the corresponding image patch $P_I;(x, y)$ for a hypothesized position (x, y). [ARS06]

maps obtained from all template patches will be combined to get the current person position, which obtained the minimal sum (vote maps actually measure dissimilarity between patches). While this approach has a drawback that an occlusion affecting even a single patch may contribute a high value to the sum at the correct position, resulting in a wrong estimate. Instead, a a LMedS-type estimator proposed in [ARS06] will be used to achieve a robust tracking result.

4.2.4.2 Person_tracker Package

In *Person Tracker* module we create one ROS package: `person_tracker`, which comes with one ROS node. Figure 4.16 displays the computation graph of `person_tracker` node. Figure 4.5 shows an example of tracking based on detection.

Figure 4.16: ROS Topics and Messages in `person_tracker` node

Implementation Details

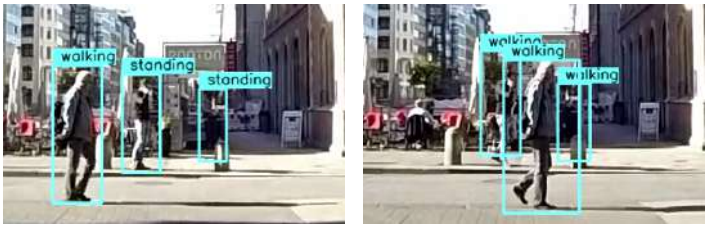
In the implementation, some problems will often occur, which could be partly solved by tuning some parameters described below. The first problem is, when two agents have an intersection, one person's bounding box will be brought away by the other. As shown in Figure 4.17, after intersection between person 2 and 3, the bounding box of person 2 drifted away. A similar situation happens in Figure 4.18, where the bounding box of person 7 didn't follow the person. After several attempts, we extend the threshold for tracking termination and set ratios both for false negative and false positive to 0.1, as shown in Figure 4.19, after intersection, the bounding box of person 2 is still at a reasonable location.

(a) Befor intersection. (b) After intersection

Figure 4.17: Demo scenario for person intersection

(a) Modified: befor intersection (b) Modified: after intersection

Figure 4.18: Demo scenario for person bounding box lagging



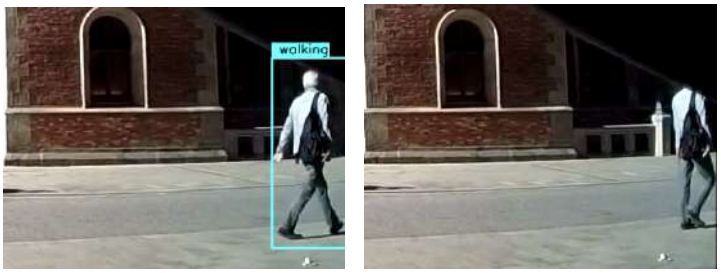
(a) Modified: befor intersection (b) Modified: after intersection

Figure 4.19: Demo scenario for person intersection after modification

The second problem happens when it comes to the boundary. As shown in Figure 4.20, after the person disappeared in the scene, an invalid bounding box still appeared. Then we add a boundary check to the framework, which checks if the bounding box has already intersected with the boundary, if so, the tracking of that person is terminated.

(a) Before intersection with boundary (b) After intersection with boundary

Figure 4.20: Demo scenario for boundary check befores modification



(a) Modified: before intersection with boundary (b) Modified: after intersection with boundary

Figure 4.21: Demo scenario for boundary check after modification

APIs Summary

- Subscriber
  Topic: `/person_detector/detection_info`
  Message type: `message_repository/FrameInfo`
  Message: detection list of persons with frame ID, image frame, person bounding box and detection confidence
- Publisher
  Topic: `/person_tracker/tracking_info`
  Message type: `message_repository/FrameInfo`
  Message: tracking list of persons with frame ID, image frame, person bounding box, person ID and tracking confidence
  `-----`
  Topic: `/person_tracker/track_render_image`
  Message type: `sensor_msgs/Image`
  Message: tracking image message rendered with person ID, bounding box and tracking confidence
- Main Paramsters
  ∼`detection_topic`: set topic for detection info subscriber
  ∼`sotracker_type`: specify the single-object-tracker type, default: `"fragment tracker"`
  ∼`feature_type`: specify feature type, default: `"FHOG"`
- Main Flags
  `--visual_tracking`: set ture to display image rendered with tracking bounding box and person ID
  `--write_video`: set true to write out video with tracking image
  `--write_track_file`: set true to write tracking list to file

4.2.5 Feature Extractor

Now we have obtained a list of persons with unique IDs. The next step is to extract the pose data based on the cropped image of each person. Since this thesis focuses on human pose, the *Feature Extractor* module equals *Pose Estimator*. While for future development, other features can also be extracted and integrated into the pipeline. In this module, two ROS packages are created for pose estimation: `openpose_ros` package and `pose_3d_ros` package.

4.2.5.1 OpenPose Library

This package is built upon the open source library — *OpenPose*[10] by Carnegie Mellon University, which is based on the works [CSWS17, SJMS17, WRKS16]. *OpenPose* represents the first real-time multi-person system to jointly detect human body, hand, facial, and foot keypoints (in total 135 keypoints) on single images using OpenCV[11] and Caffe[12] (also available in other frameworks, e.g. Tensorflow[13] and Torch[14]). An effect demonstration of *OpenPose* is displayed in Figure 4.22.

In our pipeline, we utilize the its pretrained 2D COCO body model (18 body parts), with the annotation and body joints shown in Figure 4.23, while figure 4.24 shows an example of estimated pose data, which is an array with the format $[..., x_i, y_i, confidence_i, ...,]$ of size 54 ($= 18 \times 3$).

[10] `https://github.com/CMU-Perceptual-Computing-Lab/openpose`
[11] `https://opencv.org/`
[12] `http://caffe.berkeleyvision.org/`
[13] `https://www.tensorflow.org/`
[14] `http://torch.ch/`

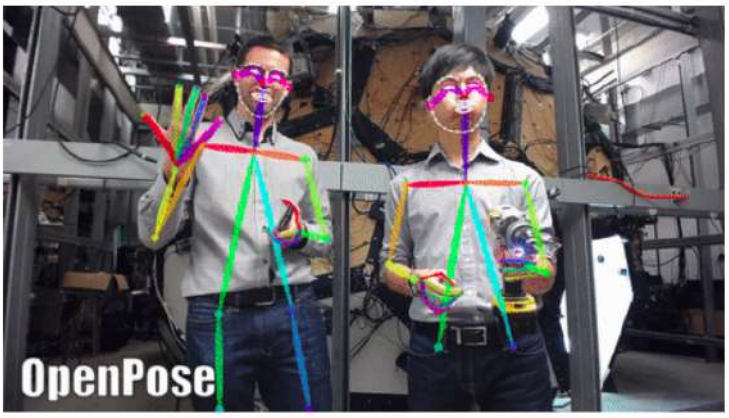
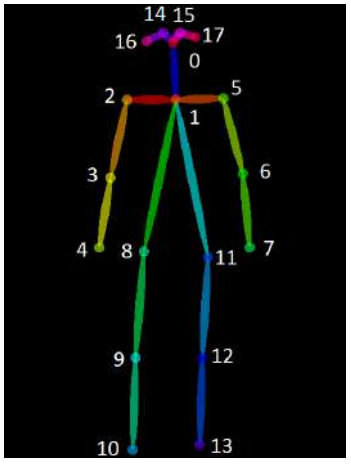


Figure 4.22: A demo of *OpenPose*[10].



(a) COCO body joints ordering.

```
{0,  "Nose"},
{1,  "Neck"},
{2,  "RShoulder"},
{3,  "RElbow"},
{4,  "RWrist"},
{5,  "LShoulder"},
{6,  "LElbow"},
{7,  "LWrist"},
{8,  "RHip"},
{9,  "RKnee"},
{10, "RAnkle"},
{11, "LHip"},
{12, "LKnee"},
{13, "LAnkle"},
{14, "REye"},
{15, "LEye"},
{16, "REar"},
{17, "LEar"},
{18, "Background"}
```

(b) COCO body parts map

Figure 4.23: COCO body model.

```
{
  "version":1.0,
  "people":[
    {
      "pose_keypoints":[
123.908,48.905,0.121667,88.0453,52.8315,0.34902,91.2871,58.6983,0.39524,
      ],
      "face_keypoints":[

      ],
      "hand_left_keypoints":[

      ],
      "hand_right_keypoints":[

      ]
    }
  ]
}
```

Figure 4.24: An example of pose data output

4.2.5.2 Openpose_ros Package

This package contains one ROS node — `openpose_ros`. The computation graph of `openpose_ros` node is shown in Figure 4.25.

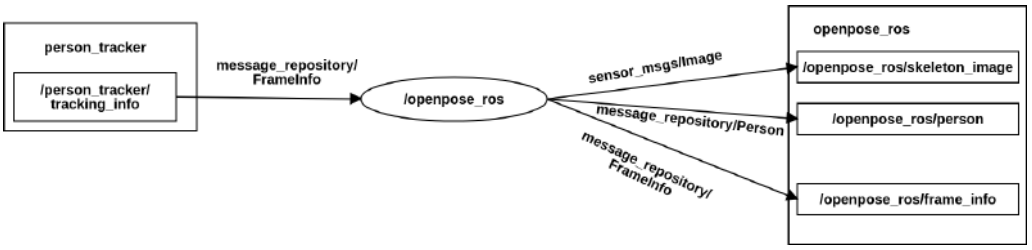


Figure 4.25: ROS Topics and Messages of `openpose_ros` node

Implementation Details

We first convert the image message contained in the frame info message from `/person_tracker` to OpenCV image, then based on person bounding box, we get the cropped images of each person. After that the cropped images will be fed into the Caffe model to get the estimated pose data for each person detected in the scene. Here we can either publish single person message (`message_repository/Person`) once each of them has done pose estimation or push them back in the frame info message (`message_repository/FrameInfo`) and send them all at once. For the transporting of pose data, we utilize `/geometry_msgs/Point` as its message type, which has the data structure listed below:

```
# This contains the position of a point in free space

float64    x
float64    y
float64    z
```

Where x holdes the value of joint x coordinate, y holds the value of joint y coordinate, while z holds the value of joint confidence (in the case of 3D pose, z corresponds to the depth of the joint).

APIs Summary

- Subscriber
  Topic: `/person_tracker/tracking_info`
  Message type: `message_repository/FrameInfo`
  Message: tracking list of persons with frame ID, image frame, person bounding box, person ID and tracking confidence

- Publisher
  Topic: `/openpose_ros/skeleton_image`
  Message type: `sensor_msgs/Image`
  Message: skeleton-rendered image message
  `-----`
  Topic: `/openpose_ros/frame_info`
  Message type: `message_repository/FrameInfo`
  Message: frame info with frame ID, image frame, person ID, bounding box, pose keypoints and confidence
  `-----`
  Topic: `/openpose_ros/person`
  Message type: `message_repository/Person`
  Message: single person message with frame ID, person ID, bounding box, pose keypoints and confidence

- Main Parameters
  ∼`tracking_info_topic`: set topic for tracking list subscription
  ∼`pose_model`: body model to be used, default: `"COCO"`

- Main Flags
  `--publish_person`: set true to publish single person message
  `--visual_skeleton`: set true to display skeleton-rendered image
  `--save_keypoints`: set true to save keypoints in json format
  `--save_skeleton_image`: set ture to save skeleton image
  `--write_video`: set true to write video with skeleton-rendered image

4.2.5.3 Stacked Hourglass Network

Another package in this module is a 3D pose estimator, which is based on the framework from [ZHS+17]. As mentioned in Section 3.3.2, the architecture of the framework consists of a 2D pose estimation module which is imported from the state-of-art hourglass architecture (shown in Figure 4.26) in [NYD16] and a depth regression module.

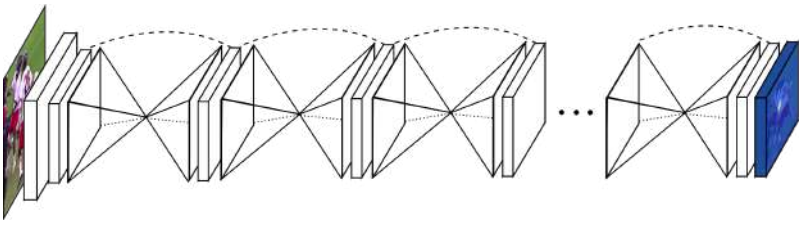


Figure 4.26: Stacked hourglass network for pose estimation consists of multiple stacked hourglass modules which allow for repeated bottom-up, top-down inference. [NYD16]

4.2.5.4 Pose_3d_ros Package

Similar to `/openpose_ros`, `/pose_3d_ros` package has one node. Figure 4.27 illustrates the computation graph of `pose_3d_ros` node.

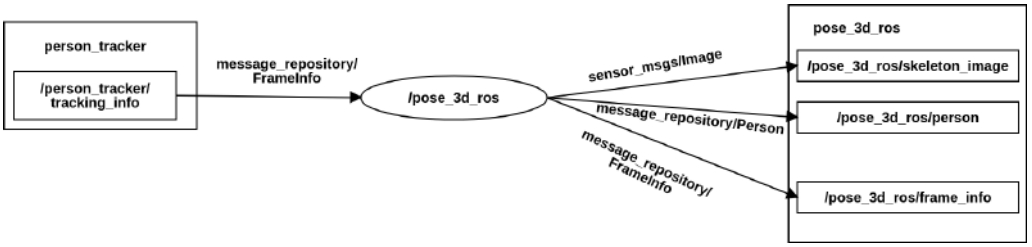


Figure 4.27: ROS Topics and Messages of `pose_3d_ros` node

Implementation Details

We utilized the PyTorch implementation of this approach with the pretrained model from the GitHub repository of the author[15]. The network takes a subject-centered RGB image which is further resized and zero-padded to a 256×256 patch as input and returns a 16×3 matrix that contains the 3D coordinates of the 16 body joints. The pose estimation result $\hat{\mathbf{P}}_{3D} \in \mathbf{R}^{J \times 3}$ of J body joints is composed of the 2D joints $\hat{\mathbf{P}}_{2D} \in \mathbf{R}^{J \times 2}$ (peak locations on the heat maps) and the depth values $\hat{\mathbf{P}}_{dep} \in \mathbf{R}^{J \times 1}$ in corresponding scales. A sample visualization of the estimated pose of a subject sitting in the chair is displayed in Figure 4.28 and configuration of body joints is shown in Figure 4.29.

[15] `https://github.com/xingyizhou/pytorch-pose-hg-3d`

(a) resized and zero-padded image patch (b) estimated 3D pose

Figure 4.28: A visualization of the estimated 3D pose of a sitting subject from the NTU RGB+D dataset [SLNW16]. The image displayed is resized and zero-padded to a 256×256 patch. The 2D component of the original estimated pose $\hat{\mathbf{P}}_{2D}$ is represented in the corresponding coordinate system of $[0, 256] \times [0, 256]$ and the depth value $\hat{\mathbf{P}}_{dep}$ is estimated in corresponding scale. Therefore, $\hat{\mathbf{P}}_{3D}$ is a "pseudo" 3D skeletal representation in a "3D image coordinate system".

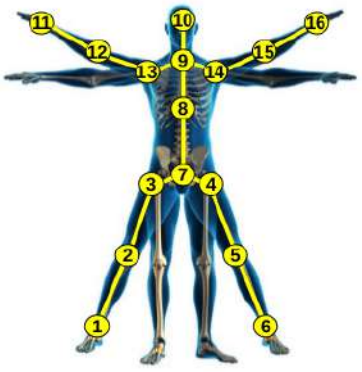


Figure 4.29: Configuration of 16 body joints in the 3D pose estimator proposed by [ZHS+17]. The labels of the 16 joints are: 1-right ankle, 2-right knee, 3-right hip, 4-left hip, 5-left knee, 6-left ankle, 7-spline base, 8-middle of spine, 9-neck, 10-head, 11-right hand, 12-right elbow, 13-right shoulder, 14-left shoulder, 15-left elbow, 16-left hand.

Since this 3D pose estimator works only for single person, we still feed the cropped image of each person into the PyTorch model and get the pose data, which is a numpy array[16] of shape $(16, 3)$. Then we either publish the single person message together with the `frame_id`, `person_id`, `image_frame`, `bbox` as well as `person_pose` or put them in one frame info message and publish them all at once.

APIs Summary

- Subscriber
  Topic: `/person_tracker/tracking_info`
  Message type: `message_repository/FrameInfo` Message: tracking list of persons with frame ID, image frame, person bounding box, person ID and tracking confidence
- Publisher
  Topic: `/pose_3d_ros/skeleton_image`

[16] `https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.array.html`

Message type: `sensor_msgs/Image`
Message: skeleton-rendered image message
`-----`
Topic: `/pose_3d_ros/frame_info`
Message type: `message_repository/FrameInfo`
Message: frame info with frame ID, image frame, person ID, bounding box, pose keypoints and confidence
`-----`
Topic: `/pose_3d_ros/person`
Message type: `message_repository/Person` Message: single person message with frame ID, person ID, bounding box, pose keypoints and confidence

- Main Parameters
  ∼`tracking_info_topic`: set topic for tracking list subscription
  ∼`pose_model`: body model to be used
- Main Flags
  `--publish_person`: set true to publish single person message
  `--save_keypoints`: set true to save keypoints
  `--save_skeleton_image`: set ture to save skeleton image

4.2.6 Data Manager

After pose estimation, *Data Manager* takes the responsibility to process data (e.g. interpolation, smoothing, normalization, etc.), so as to prepare the pose tensor for *Action Classifier*.

4.2.6.1 Formation of Pose Tensor

In order to classify a video sequence to an action category, a convolutional neural network (PoseNet, see Section 4.2.7) is utilized to learn the spatio-temporal features of 2D/3D poses from the entire sequence. The spatial information is the skeletal structure of body joints that intuitively represents a human posture in one video frame. The temporal information is the dynamics of human postures contained in the trajectory of body joints in an action sequence. For the data-driven learning based on convolutional kernels instead of recurrent networks, it is essential that both the spatial structure of body joints and the temporal evolution of skeletons are encoded in a compact manner. Accordingly, the pose tensor is formed in a compact structure with 2D/3D skeletal information of an action sequence. The topological ordering and three-channel-structure of pose tensor are introduced in the following.

Topological Ordering

Following the work of [LSXW16], a tree-structured traversal is designed to define a topological ordering of the human body joints. As is illustrated in Figure 4.30, a cyclic path is constructed by a traversal with repetition along the tree structure. Starting from the root node of joint *spine base* (joint 7), the path goes forward until a leaf node is reached and then moves backward back to the nearest node with at least two children. The forward and backward pass are conducted repeatedly in this manner until every leaf node in the tree is reached. In this case, every limb (or edge in the tree structure) is passed exactly twice, once in the forward pass from parent to child and once backward from child to parent. The jth joint (a node in the tree structure) that connects n_j limbs is visited exactly n_j times in this cyclic path, except for joint 7 (the root node) that is visited $n_7 + 1$ times. For example, the five external joints 10, 11, 16, 1, 6 (*head, right hand, left hand, right ankle, left ankle*) are visited only once. The internal joints such as 12, 15 (*right elbow, left elbow*)

that connect two body limbs are visited two times. The important junction 9 (*neck*) that connects four body limbs is visited four times. In this way, neighborhood information in the body configuration is preserved, as a body joint always appears with its adjacencies in the node sequence yielded by this topological ordering [Lin18].

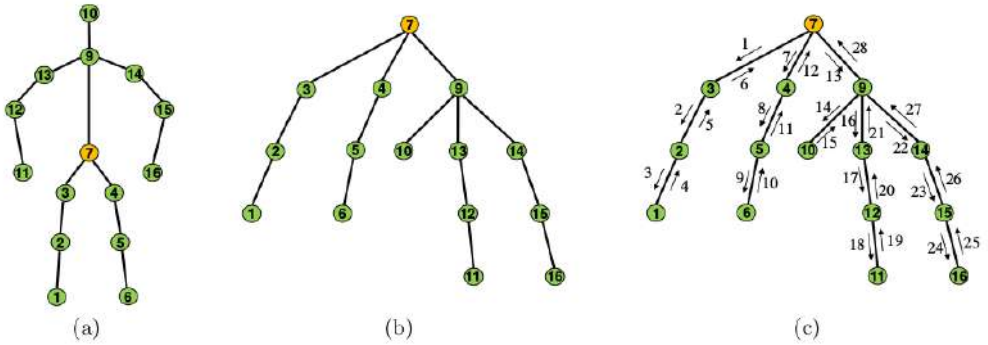


Figure 4.30: (a) The body configuration of 15 joints (joint 8 is omitted). (b) The full body is unfolded into a tree structure, with joint 7 (*spine base*) as the root node (c) The topological ordering is defined by a cyclic path along the tree structure. Each limb is visited twice in the bidirectional traverse with repetition and each joint always appears with its neighbors in the sequence. The generated node sequence is 7-3-2-1-2-3-7-4-5-6-5- 4-7-9-10-9-13-12-11-12-13-9-14-15-16-15-14-9-7. [Lin18]

Similar to [BWM17], the first channel of pose tensor is constructed by arranging the 3D coordinates of body joints from all video frames in a compact manner, as is shown in Figure 4.31. We first concatenate 3D coordinates x, y and z of joints from one frame as a row vector in the topological ordering designed above. Then the 3D coordinates from different frames in an action sequence are stacked row-wisely. We denote the length of the node sequence of topological ordering as L and the number of frames in an sequence as T.

After the concatenation and row-wise stacking, we end up with a 2D matrix $\mathbf{X} \in \mathbf{R}^{T \times 3L}$. In order to deal with videos of varying length of frames, the skeleton data matrix $\mathbf{X}$ is regarded as an image and its first dimension is normalized to a fixed value K by a basic image resizing operation with bi-cubic interpolation. We acquire the normalized matrix of skeleton data $\tilde{\mathbf{X}} \in \mathbf{R}^{K \times 3L}$. We set $K = 10$.

Three-Channel-Structure

As illustrated in Figure 4.31, the normalized K-frame matrix of raw 3D coordinates of joints $\tilde{\mathbf{X}}$ constitutes the first channel of the 3D pose tensor. The temporal evolution of pose is regarded as a continuous and differentiable function of body joints over time. Additional kinematic features can be added by computing the derivatives of joint positions. Accordingly, the second channel is composed by the first derivatives of the 3D coordinates, which represent the velocities of body joints between two consecutive frames. The third channel consists of the second derivatives, which describe the acceleration of body joints. In order to recover the actual velocity and acceleration in the original action sequence of T frames. We compute the temporal step s between two frames in the resized K-frame matrix by

$$s = floor(\frac{T}{K}) \tag{4.2}$$

Then we denote the x-component of the raw 3D coordinates, its first derivative and the

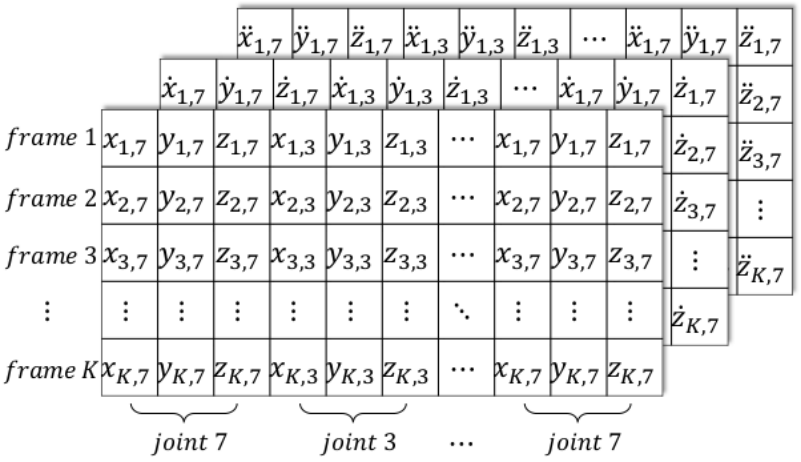


Figure 4.31: The 3D pose tensor $\mathbf{T} \in \mathbf{R}^{K \times 3L \times 3}$: the first dimension is the frame index, the second dimension is the x, y and z coordinates of all the body joints concatenated in the topological ordering designed in Figure 4.30, the third dimension contains raw coordinates, velocities, accelerations as the three channels. [Lin18]

second derivative of the jth joint in the kth frame as $x_{k,j}$, $\dot{x}_{k,j}$ and $\ddot{x}_{k,j}$. The derivatives can be derived by

$$\dot{x}_{k,j} = \begin{cases} 0, & k = 1 \\ \frac{x_{k,j} - x_{k-1,j}}{s}, & k \geq 2 \end{cases} \tag{4.3}$$

$$\ddot{x}_{k,j} = \begin{cases} 0, & k = 1 \\ \frac{\dot{x}_{k,j} - \dot{x}_{k-1,j}}{s}, & k \geq 2 \end{cases} \tag{4.4}$$

Now, for each action sequence, we have formed the uniform-sized pose tensor $\mathbf{T} \in \mathbf{R}^{K \times 3L \times 3}$.

4.2.6.2 Pose Processing

Often is the case that the pose data obtained from pose estimator come with some flaws, such as missing joint (occulusion), outliers and so on. Therefore *Data Manager* will perform pose spatial interpolation, temporal smoothing as well as normalization before the pose tensor is sent for action prediction.

Temporal Smoothing of Pose

Videos contain rich motion information that covers the smooth movement of human joints. Because of this movement, some invisible joints become visible in continuous frames or some visible joints become invisible due to self or object occlusion. The interpolation scheme is based on the assumption that the movement of joints is linear for small temporal range of frames. Therefore, by utilizing the available joints in small temporal range of frames F, the position of missing joint positions can be interpolated from small temporally neighboured frames. A P-Spline interpolation[17] is used to estimate the location of missing joint positions. The interpolation uses valid joint positions in temporal range of F frames for estimating missing joint positions. Missing joint position for a particular joint

[17] https://en.wikipedia.org/wiki/Smoothing_spline

j is interpolated by using the `scipy.interpolate.UnivariateSpline` class in `Scipy` library[18]. Figure 4.32 displays how P-Spline smoothing recovers a single missing joint along the timeline of 10 frames.

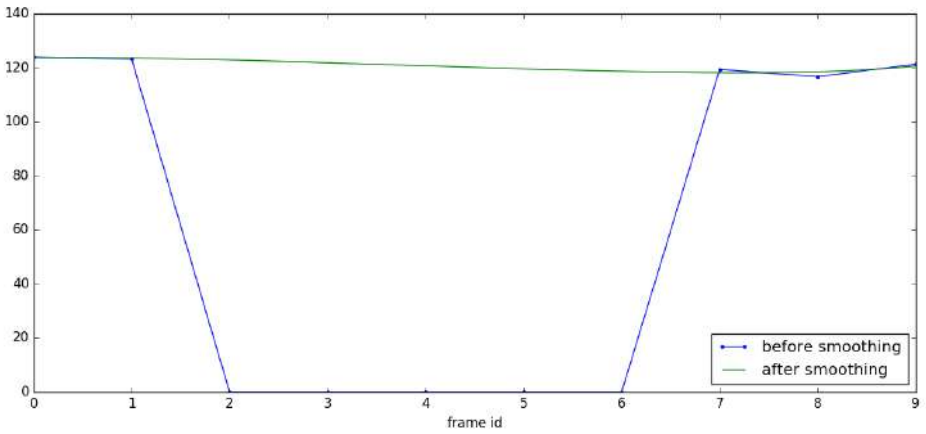


Figure 4.32: An example of temporal P-Spline smoothing. Before smoothing (blue), the first 10 frames of the joint x position are: [123.908, 123.259, 0., 0., 0., 0., 0., 119.357 116.73, 121.292], After smoothing (green), the values are: [123.71374527 123.55038209, 122.86919667, 121.85577574, 120.69570604, 119.5745743, 118.67796725, 118.19147164, 118.47829254, 120.61210846]. The missing points in frame 2 to 6 have been smoothingly interpolated.

Spatial Interpolation of Pose

However, temporal interpolation has its limitation for long-term occluded human joints. If some joints are not detected for a long temporal range, the linear motion assumption for temporal interpolation will not be valid any more. Therefore, for the cases without information of the corresponding missing joint positions in some other frames, an additional interpolation is carried out. As proposed in [Kha18], this interpolation uses the available joint positions in the same frame to estimate missing joint positions in that frame. The idea is based on the observation that the location of human joints are strongly correlated with each other especially the neighbouring joints, such as shoulder and elbow joints. Inspired by the work in [MA10], neighbourhood relationships of joints are utilized to vote for possible location of missing joints.

A polynomial curve fitting model is formulated that models the spatial relationship of neighboured joints. This model is learned from the varied video datasets with ground truth poses. Figure 4.33 shows the statistical polynomial curve fitting model learned for the estimation of location of *Left Ankle* joint. 4.33(a) shows the correlation of normalized x-coordinate of *Left Hip* joint and normalized x-distance between *Left Hip* and *Left Ankle* joint. The blue dots denote the data points for normalized x-distance between *Left Hip* and *Left Ankle* with respect to normalized x-coordinate of *Left Hip* joint, for all the poses in dataset. The red line shows the curve fitting model learned from the data points. The similar curve fitting model is learned from the data points of the normalized y-coordinate of *Left Hip* joint and normalized y-distance between *Left Hip* and *Left Ankle* joint as shown in Figure 4.33(b).

As direct neighboured joints have strong correlation, the whole body is divided into 5

[18] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html`

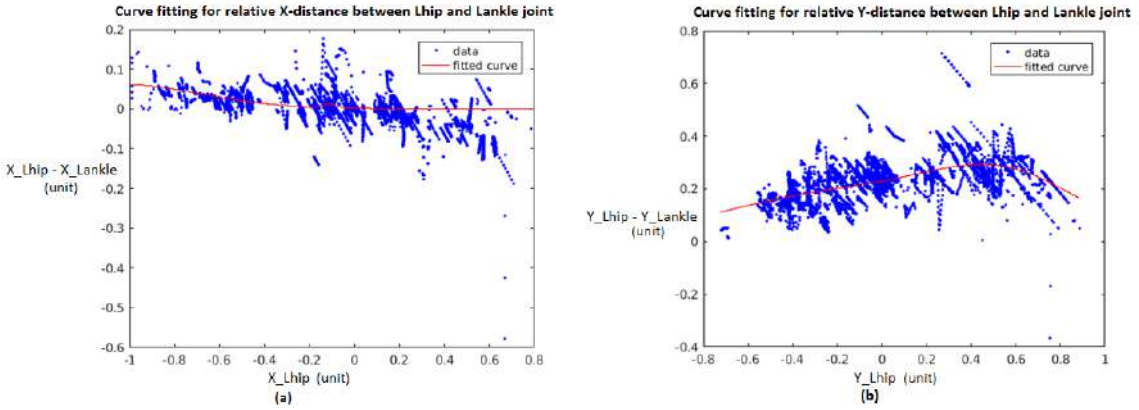


Figure 4.33: Statistical curve fitting model for relative relative joint positions of *Left Ankle* and *Left Hip* joints. (a) Polynomial Curve fitting model for estimation of x-coordinate of *Left Ankle* joint using *Left Hip* joint (b) Polynomial Curve fitting model for estimation of y-coordinate of *Left Ankle* joint using *Left Hip joint*. [Kha18]

body parts keeping their tight neighbourhood relationships intact as shown in Figure 4.34. Body part 1 to 4 has tight spatial neighbourhood relationship but body part 5 has loose neighbourhood relationship. For a missing joint position within frame, firstly all the available joint positions in the corresponding body part of tight neighbourhood relationship (part 1 to part 4) are selected to estimate the missing joint position. If no joint position in corresponding part is available, then joints from body part 5 are selected for the estimation of missing upper body joints. For all other cases, all the available joint positions in that frame are selected for estimation of missing joint. Each available joint in respective body part votes for the missing joint position and all the votes of selected joints are averaged to find the final estimation of missing joint.

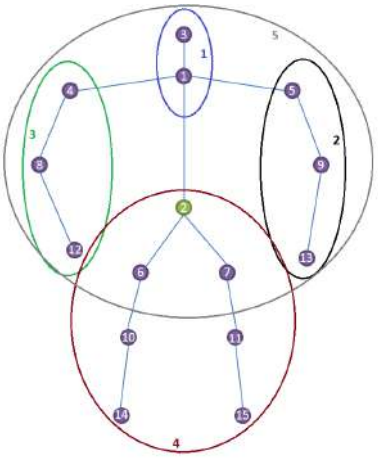


Figure 4.34: Configuration of 5 body parts for spatial interpolation of missing joint positions [Kha18]

Figure 4.35 shows the effects of pose spatial interpolation and temporal smoothing.

Pose Normalization

The 2D joint positions of humans are simply the image coordinates (x, y) of the corresponding joints. These image coordinates are sensitive to camera perspectives and image

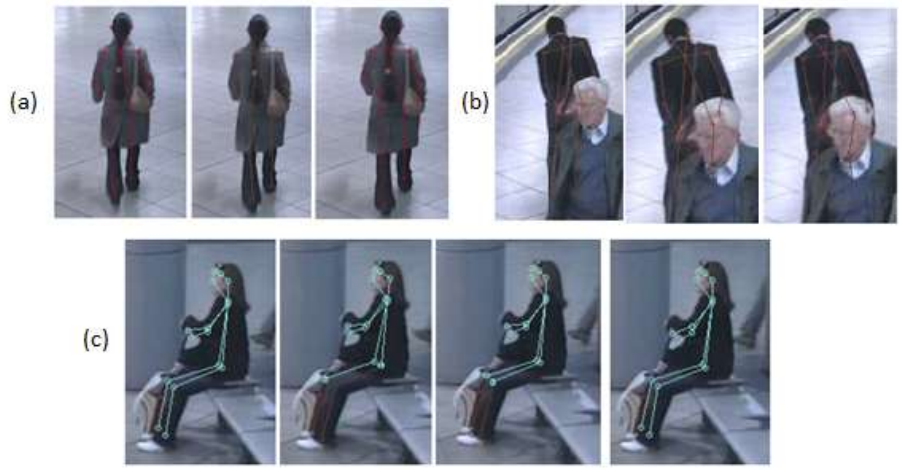


Figure 4.35: Complete body pose after temporal and spatial interpolation (a) Missing pose in some frames estimated by temporal interpolation (b) Occluded joint positions estimated by spatial interpolation (c) Missing joint positions in some frame estimated by temporal interpolation [Kha18]

resolution. Thus a normalization is required, which keeps all the poses to be of similar size and to be at the centre of the image. As given in Eqution 4.5, all n joint positions of the body are normalized with respect to image coordinate system in the range of $[-1, 1]$, to get rid of the dependency of pose to video resolution [Kha18].

$$\hat{P}_{i(x,y)} = 2 \times \frac{OP_{i(x,y)}}{(I_w, I_h)} - 1, \quad \forall i = 1, \ldots, n \tag{4.5}$$

where I_w and I_h correspond to width and height of frame respectively. $OP_{(x,y)}$ is the joint positions in original image coordinate. These normalized joint positions $\hat{P}_{(x,y)}$ are then scaled with respect to torso length as follows:

$$\bar{P}_{i(x,y)} = \frac{\hat{P}_{i(x,y)}}{d}, \quad \forall i = 1, \ldots, n \tag{4.6}$$

with,

$$d = \sqrt{(\hat{x}_{neck} - \hat{x}_{belly})^2 + (\hat{y}_{neck} - \hat{y}_{belly})^2} \tag{4.7}$$

Finally, these rescaled joints $\bar{P}_{(x,y)}$ are shifted at the center of torso, given below:

$$\hat{P}_{torso(x,y)} = \frac{\bar{P}_{neck(x,y)} + \bar{P}_{belly(x,y)}}{2} \tag{4.8}$$

$$P_{i(x,y)} = \bar{P}_{i(x,y)} - \bar{P}_{torso(x,y)}, \quad \forall i = 1, \ldots, n \tag{4.9}$$

$P_{i(x,y)}$ is the normalized, scaled and shifted joint positions for all joints, as shown in Figure 4.36. These normalized joint positions $P_{i(x,y)}$ are then used to form pose tensor as described above.

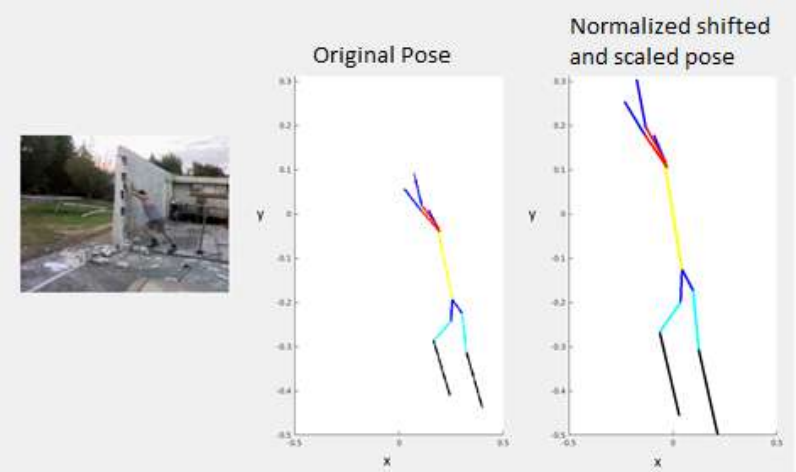


Figure 4.36: Pose Normalization [Kha18]

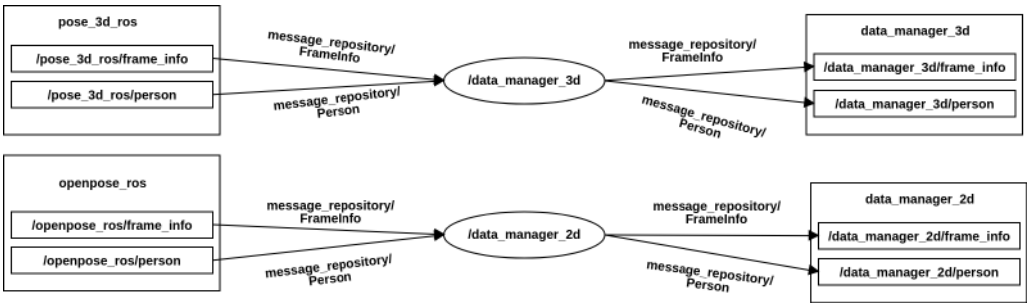


Figure 4.37: ROS Topics and Messages in `data_manager_2d` and `data_manager_3d` nodes

4.2.6.3 Data_manager Package

In this module we created one ROS package — *data_manager* with two nodes: `data_manager_2d` for 2D pose and `data_manager_3d` for 3D pose. Figure 4.37 shows the computaion graph of *data_manager* package.

Implementation Details

Since these two nodes have similar structures, we take `data_manager_3d` as example. First we extract person ID and person pose information from the tracking info message. Then we will determin if the person has already appeared before. In order to store the ID and pose data for each person, we create a pose pool as container, its structure is shown in Table 4.1.

APIs Summary

ROS node: `data_manager_3d`

- Subscriber
  Topic: `/pose_3d_ros/frame_info`
  Message type: `message_repository/FrameInfo`
  Message: frame info with frame ID, image frame, person ID, bounding box, pose keypoints and confidence
  `-----`
  Topic: `/pose_3d_ros/person`
  Message type: `message_repository/Person`
  Message: single person message with frame ID, person ID, bounding box, pose keypoints and confidence

Dict	Pose Pool			
person id (key)	0	1	2	...
pose_pool (item member)	... frame i: [..., x_i_n, y_i_n, z_i_n, ...] frame j: [..., x_j_n, y_j_n, z_j_n, ...] ...	...	...	...
sample_id (item member)	[..., i, j, ...]	...	...	...

Table 4.1: **Data structure of Pose Pool**. Pose Pool is no more than a dictionary (Python) or map (C++). Person ID is used as the key, each time `data_manager_3d` receives the person pose from *Pose Estimator*, it will first check if the person has already a record in the Pose Pool. If not, the Pose Pool will create a block for the person, otherweise Pose Pool will update the person's `pose_pool`. The `pose_pool` of each person is a concatenated array of pose data from different frames, and the pose data of each frame is an array of (x, y, z) of all joints in the node sequence (e.g. in Figure 4.30). While another item member — `sample_id` determines which frame should be handled. As default in our pipeline, we will process every single frame, so the `sample_id` list would be: [0, 1, 2 , 3, ...], at the end of the pose processing, the next sample id that should be handled will be generated and be appended to `sample_id` list. Once a person's `pose_pool` has achieved a predefined number K (here we set K = 10), we will generate the pose tensor from the K frames with pose processing as mentioned above. Then we store the pose tensor in the `Person` message or gather all the persons in one `FrameInfo` message and publish it. At last we remove the first frame in the person's `pose_pool`, and concatenate the coming pose frame to the end of it, so as to form a new pose tensor, while reserving previous $K - 1$ pose frames.

- Publisher
  Topic: `/data_manager_3d/person`
  Message type: `message_repository/Person`
  Message: single person message with person ID and pose tensor
  `-----`
  Topic: `/data_manager_3d/frame_info`
  Message type: `message_repository/FrameInfo`
  Message: frame info message with all persons with their own ID and pose tensor
- Main Parameters
  ∼`frame_info_topic`: set topic for frame info message subscriber
  ∼`person_topic`: set topic for person message susbcriber
  ∼`tensor_length`: define the number of frames in one pose tensor, default: 10
  ∼`tensor_offset`: define the start frame of pose tensor, default: 0
  ∼`tensor_stride`: define the stride between two consecutive to-be-handled frames, default: 1
  ∼`node_sequence`: specify the traversal sequnce

4.2.7 Action Predictor

After the generation of pose tensor, the last step we will take is to feed the pose tensor to the PoseNet in *Action Predictor*.

4.2.7.1 Network Structure

In our pipeline, two PoseNets will be utilized for action classification, one for 2D pose and the other for 3D pose.

PoseNet for 2D Pose

A Pose ConvNet is proposed by [Kha18], which is trained end-to-end by taking pose tensor as input and action scores as output. As shown in Figure 4.38, the Pose ConvNet architecture is a shallow network because pose tensors contain highly compact data and consists of high-level features.

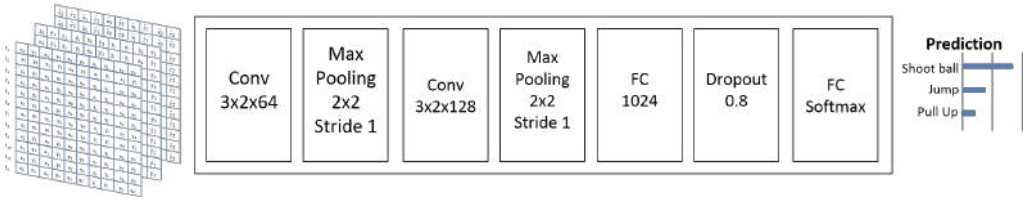


Figure 4.38: Architecture of Pose ConvNet [Kha18]

PoseNet for 3D Pose

Another PoseNet for 3D pose based action classification is proposed in the work [Lin18], which consists of two convolutional layers along with two max pooling layers and two fully connected layers. The architecture is illustrated in Figure 4.39.

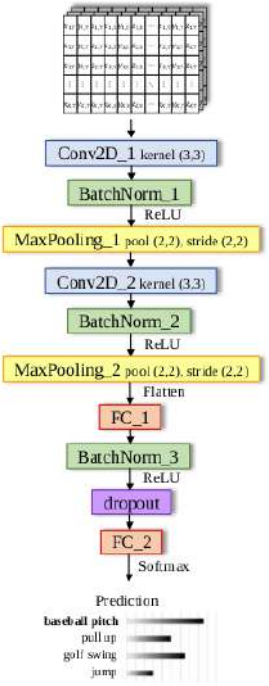


Figure 4.39: **Architecture of PoseNet**. A CNN of two Conv layers, two Max Pooling layers and two FC layers. Hyperparameters such as number of kernels in Conv layers and size of FC layers are determined according the scale of the dataset. [Lin18]

4.2.7.2 Action Proposal

Now we will get one action label per frame (except for the first K = 10 frames), but what we really need is the beginning and ending of some action. In our pipeline, we generate the

pose tensor based on 10 consecutive frames, which is unavoidably noisy, as the completion of an action usually takes certain mount of time (frames), while in such a short period of time, some misunderstanding will probably appear. An example is illustrated in Figure 4.40, if we just take the prediction label of each pose tensor, we will get a very noisy and less reasonable result, as denoted in green in bottom diagram. Inspired by the work [XZW+17], we proposed an action proposal strategy, which effectively reduces the prediction noise, as shown in Figure 4.40 bottom diagram (red).

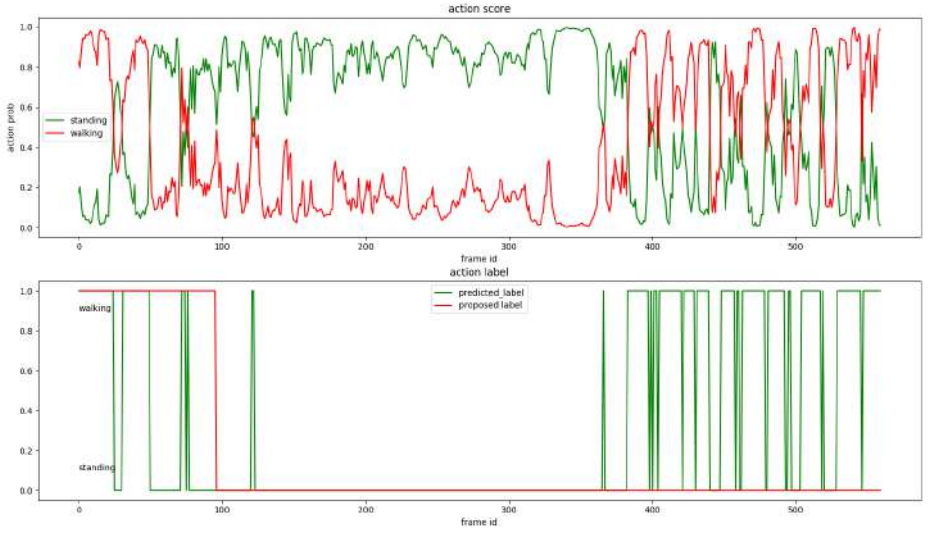


Figure 4.40: Action Classification with and without Proposal. The top diagram displays the action score of the action standing (green) and walking (red) along the timeline (frames) respectively. The bottom diagram shows the action label (action with higher scores, 0 – standing, 1 – walking) of each frame, the green one stands for the prediction based on the current pose tensor, while the red one stands for the proposed label.

Our action grouping algorithm (only for standing and walking) is illustrated as pseudo code 1. Here τ is the action threshold, which means when the frame label (predicted based on current pose tensor) is different from last proposed label, only if when the new action score is no less than τ can it be taken as valid and increase the count by 1, so as to tell the algorithm that the frame label may be true, rather than a noise or an outlier value. When the count value achieves the tolerance threshold γ, the frame label is taken as the new valid action label. However, this algorithm also comes with some drawbacks. One of them is, the parameters of τ and γ is not easy to set for all scenarios. Another drawback says, if a new reasonable action appears, it still takes at least γ frames to switch to the new action prediction, so as to show a delay for the prediction. There is always a compromise to be made between noise reduction and high sensitivity (once a change in action happens, the algorithm should resond to it as soon as possible).

4.2.7.3 Pose_net Package

Similar to `data_manager`, `pose_net` also has two nodes, one for 2D pose and the other for 3D pose. Figure 4.41 shows the computation graph of `pose_net` package.

Implementation Details

Algorithm 1: Action Grouping Algorithm

```
Result: Proposed action label
set proposedLabel = None;
set proposedLabelList = None;
set frameLabel = None;
set standingScore = 0;
set walkingScore = 0;
set τ = 0.5;
set γ = 10;
set count = 0;
while True do
    standingScore = getStandingScore();
    walkingScore = getWalkingScore();
    frameLabel = getFrameLabel();
    if first prediction then
        proposedLabel = frameLabel;
    end
    else
        if proposedLabelList[-1] == frameLabel then
            proposedLabel = frameLabel;
            count = 0;
        end
        else
            if standingScore >= tau and count >= γ then
                proposedLabel = standing;
                count = 0;
            end
            else if walkingScore >= tau and count >= γ then
                proposedLabel = walking;
                count = 0;
            end
            else
                proposedLabel = proposedLabelList[-1];
            end
        end
    end
    proposedLabel.append(proposedLabel);
end
```

In our pipeline, we have currently tested and evaluated the 3D pose, for which we have trained several models. In one of the models, we add some data jittering augmentation at the training stage, trying to compensate the instability of camera perspectives and the tracking bounding box. Another model is trained with pose tensor generated with stride 1, which works exactly as runtime process. Some results and evalutions will be further discussed in Section 5.4. Similar to the `Pose Pool` data structure, we employ an `Action Pool` container to store the proposed action label and the corresponding frame ID, so as to make the online action proposal easier. As `pose_net_2d` and `pose_net_3d` node share similar APIs, we take `pose_net_3d` as example to explain the APIs.

APIs Summary

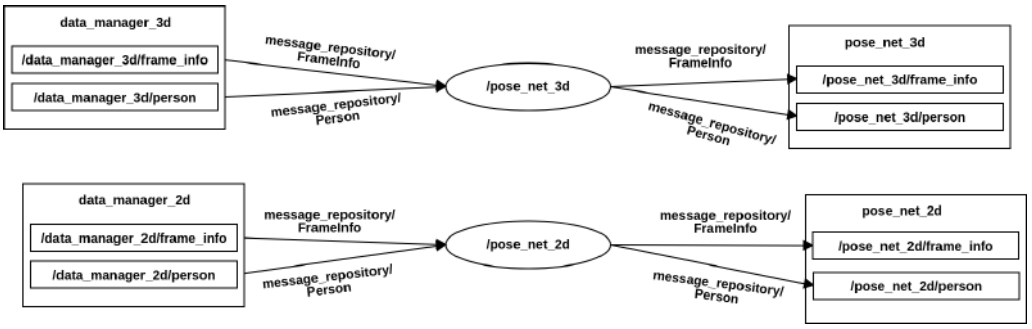


Figure 4.41: ROS Topics and Messages in `pose_net_2d` and `pose_net_3d` node

ROS node: `pose_net_3d`

- Subscriber
  Topic: `/data_manager_3d/frame_info`
  Message type: `message_repository/FrameInfo`
  Message: all persons with their ID and pose tensor
  `-----`
  Topic: `/data_manager_3d/person`
  Message type: `message_repository/Person`
  Message: single person message with person ID and pose tensor
- Publisher
  Topic: `/pose_net_3d/person`
  Message type: `message_repository/Person`
  Message: single person message with person ID and person action label
  `-----`
  Topic: `/pose_net_3d/frame_info`
  Message type: `message_repository/FrameInfo`
  Message: frame info message with all persons with their own ID and action label
- Main Parameters
  ∼`frame_info_topic`: set topic for frame info message subscriber
  ∼`person_topic`: set topic for single person message susbcriber
  ∼`pose_model`: specify the PoseNet model
- Main Flags
  `--plot_action`: set true to plot action proposal online

4.2.8 Visual Agency

The very basic idea behind this module is that, the other modules can focus on their main functions and leave the job of visualization, logging or other data IO to *Visual Agency*, which is also of great importance for testing and debugging the pipeline.

4.2.8.1 Visual_agency Package

In this package, we now have a `visual_agency` node, which subscribes to several topics in the pipeline. As shown in Figure 4.42, it basically visualizes the results, especially rendered image with different information, from different modules.

Implementation Details

Though the `visual_agency` node provides multiple options for visualizing different images, but the message on topic `/pose_net_2d/frame_info` and `/pose_net_3d/frame_info` has

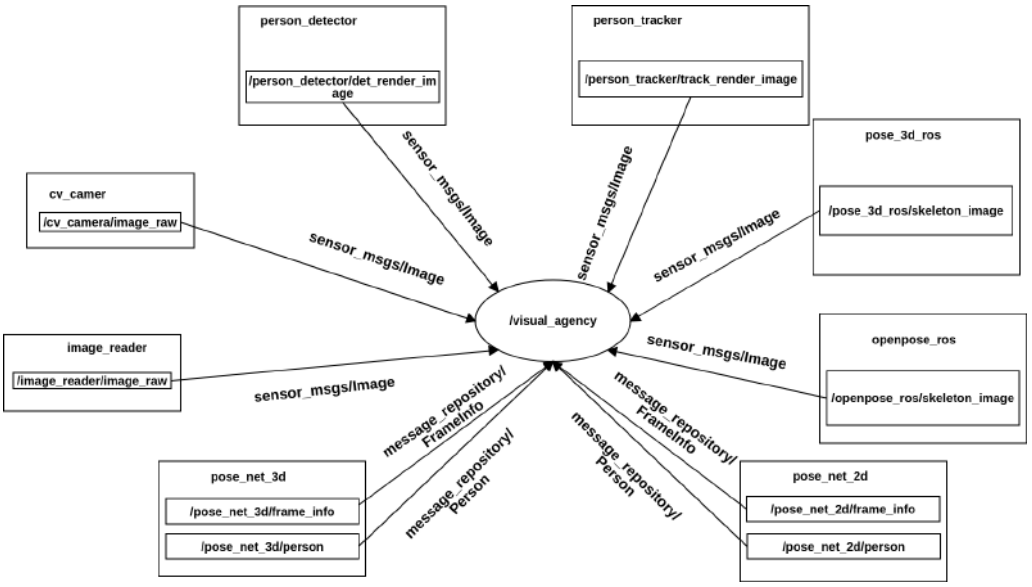


Figure 4.42: ROS Topics and Messages in `visual_agency` node

contained all the needed information that is defined in the `FrameInfo.msg`. Instead of setting different subscribers, a better option is to set different flags for showing the corresponding images or data based on the frame info message from *Visual Agency*.

APIs Summary

ROS node: `visual_agency`

- Subscriber
  Topic: `/pose_net_3d/frame_info` and
  Message types: `message_repository/FrameInfo`
  Message: all persons with frame ID, image frame, person ID, person bounding box, person action, etc. as described in message definition of `FrameInfo.msg`
  `-----`
  Topic: `/pose_net_3d/person`
  Message type: `message_repository/Person`
  Message: single person message with frame ID, person ID, person image, person action in the frame.
  `-----`
  Topic: image topics in the pipeline as shown in Figure 4.42
  Message type: `sensor_msgs/Image`
  Message: images rendered with multiple information

- Main Parameters
  ∼`frame_info_topic`: set topic for FrameInfo message subscriber
  ∼`person_topic`: set topic for Person message susbcriber
  ∼`image_topic`: set topic for image message subscriber

- Flags
  `--visual_raw_image`: visualize original raw image frame
  `--visual_detection`: visualize image rendered with detection bounding box and confidence
  `--visual_tracking`: visualize image rendered with tracking bounding box and person ID
  `--visual_action`: visualize image rendered with person action label

`--write_video`: write output video with image rendered with action
`--write_det_file`: write detection list into file
`--write_track_file`: write tracking list into file
`--write_action`: write action label into file

4.2.9 Pipeline Utilities

This module holds all other packages or modules that are utilized as auxiliary roles. Apart from the *Message Repository* introduced above, `test_n_debugger`, *3rdPartyLibs* together with `rp_utilities` consititute the rest of module *Pipeline Utilities*. `test_n_debugger` is mainly created for managing all tests (unittest[19], gtest[20]) and conducting experiments, while *3rdPartyLibs* contains the main libraries that are used in the pipeline, e.g. OpenPose, Darknet (YOLO), PersonTracker, etc. The `rp_utilities` package includes some ROS-related utilities, such as launch file, bag file and so on.

[19] `https://docs.python.org/3/library/unittest.html`
[20] `https://github.com/google/googletest`

5. Experiments and Evaluation

In this chapter, we first make a brief introduction to the hardware development platform and the datasets employed in our pipeline. Then we conduct three experiements both on groudtruth and original videos to evaluate the effectiveness and run-time performance of our pipeline. At last the experiment results will be discussed.

5.1 Hardware Platform

Our work is built mainly on two types of hardware platform: Nvidia Jetson TX2 Developer Kit[1] for pipeline development and Server equipped with GeForce GTX 1080 GPUs[2] for real-time testing.

Nvidia Jetson TX2 Developer Kit

NVIDIA Jetson TX2 is an embedded system-on-module (SoM) with dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57, 8GB 128-bit LPDDR4 and integrated 256-core Pascal GPU. Its board and module components are shown in Figure 5.1 and 5.2.

Figure 5.1: Nvidia Jetson TX2 Board

5.2 Datasets

In our pipeline, we mainly work on the dataset Joint Attention in Autonomous Driving (JAAD) [KRT16].

[1] `https://elinux.org/Jetson_TX2`
[2] `https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/`

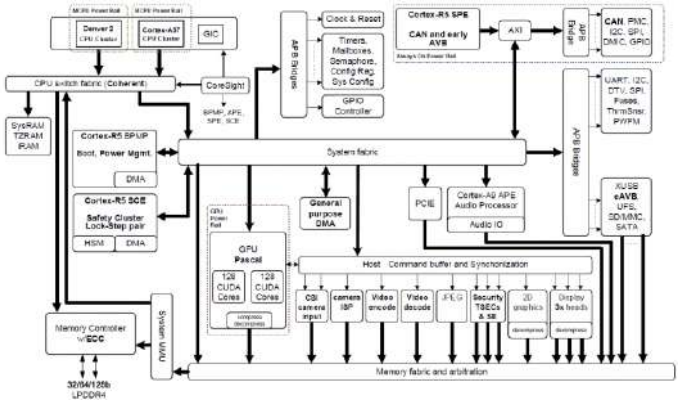


Figure 5.2: Nvidia Jetson TX2 Module Diagram

5.2.0.1 JAAD Dataset

As presented in [RKT17], JAAD dataset is a new dataset for studying joint attention in the context of autonomous driving. In particular, the focus is on pedestrian and driver behaviors at the point of crossing and factors that influence them. To this end, JAAD dataset provides an annotated collection of short video clips representing scenes typical for everyday urban driving in various weather conditions.

JAAD dataset contains 346 high-resolution video clips (most are 5-10 sec) extracted from approximately 240 hours of driving videos filmed in several locations in North America (60 clips) and Eastern Europe (286 clips) using three different high-resolution monocular cameras with a frame rate of 30 fps. The dataset comes with three types of ground truth annotations: bounding boxes for detection and tracking of pedestrians, behavioral tags indicating the state of the pedestrians and scene annotations listing the environmental contextual elements.

Bounding boxes

the dataset contains approximately 82k frames and 2.2k unique pedestrian samples comprising a total of 337k bounding boxes. Occlusion information is provided in the form of tags for each bounding box: partial occlusion (between 25 and 75% visible) and full occlusion (less than 25% visible). ∼55K (13%) of bounding boxes are tagged with partial occlusion and ∼49K (12%) with heavy occlusion.

Behavioral tags

Behavioral data and attributes are provided for 868 pedestrians. Figure 5.3 illustrates a scene of a temporal behavioral labels in the dataset.

The pedestrians' actions are categorized into 3 groups:

Precondition this refers to the state of the pedestrian prior to crossing and can be either standing, moving slow or fast.

Attention the way the pedestrian becomes aware of the approaching vehicle. These actions, depending on their duration, are looking ($>$ 1s) or glancing ($\leq$ 1s).

Response this includes the behaviors that pedestrians exhibit in response to the action of the approaching vehicle, namely, stop, clear path, slow down, speed up, hand gesture and nod.

Contextual tags

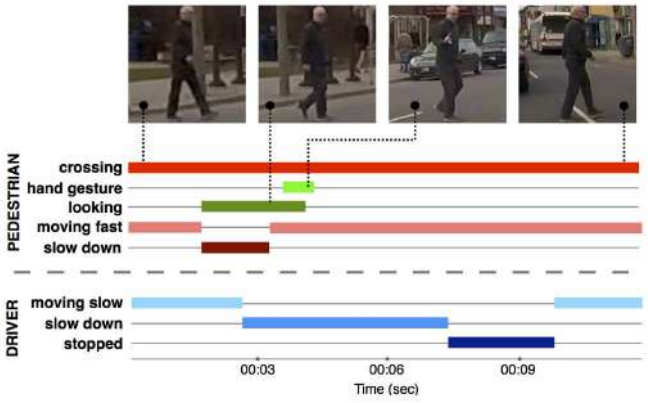


Figure 5.3: Behavioral labels with timestamps to represent the sequence of the events that took place during the crossing and observe the correspondence between the actions of the driver and the pedestrian. [RKT17]

Each frame is assigned a contextual tag that describes the scene. There are four types of contextual information:

Configuration includes the number of lanes or whether it is a parking lot/garage.

Traffic signals refers to the presence of zebra crossing, pedestrian sign, stop sign or traffic light in the scene.

Weather includes sunny, cloudy, snowy or rainy.

Time of day this tag crudely indicates the lighting conditions and can be day, afternoon or nighttime.

Figure 5.4 displays some selected pedestrians from the dataset.

Figure 5.4: A selection of images of pedestrians from the dataset. [RKT17]

5.3 Experiments

The experiments pipeline is as follows, shown in Figure 5.5:

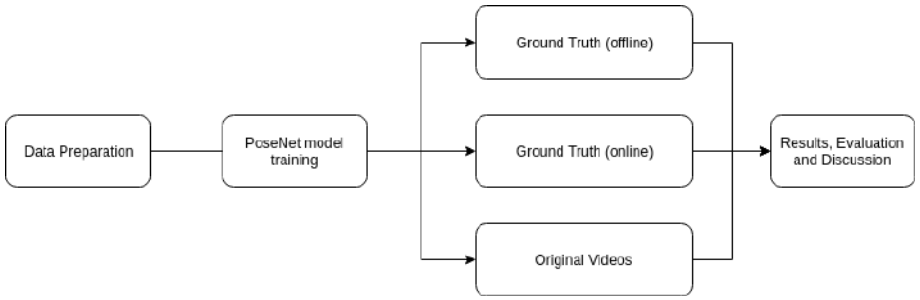


Figure 5.5: The pipeline of experiments

Data Preparation

The training of PoseNet is based on the ground truth from JAAD dataset, which contains image sequences of 698 pedestrians from different video clips. For each pedestrian image sequences, we first extract 3D pose of each frame and write the data in Hierarchical Data Format[3] (HDF5). As introduced above, we traverse the body map to form the pose tensor, in our experiments, we set the traversal sequence to [6, 2, 1, 0, 1, 2, 6, 3, 4, 5, 4, 3, 6, 8, 9, 8, 12, 11, 10, 11, 12, 8, 13, 14, 15, 14, 13, 8, 6], which stands for ['Pelvis', 'RHip', 'RKnee', 'RAnkle', 'RKnee', 'RHip', 'Pelvis', 'LHip', 'LKnee', 'LAnkle', 'LKnee', 'LHip', 'Pelvis', 'Neck', 'Head', 'Neck', 'RShoulder', 'RElbow', 'RWrist', 'RElbow', 'RShoulder', 'Neck', 'LShoulder', 'LElbow', 'LWrist', 'LElbow', 'LShoulder', 'Neck', 'Pelvis'] in the model topology shown in Figure 4.30. Then we get 698 HDF5 files, each of them contains the pose keypoints of a pedestrian with the shape of (N, M), here N is the number of frames and M stands for the number of joint positions, which, in our case, is 87 ($= 29 \times 3$, 29 is the number of traversal sequence nodes listed above, 3 stands for (x, y, z)).

PoseNet Model Training

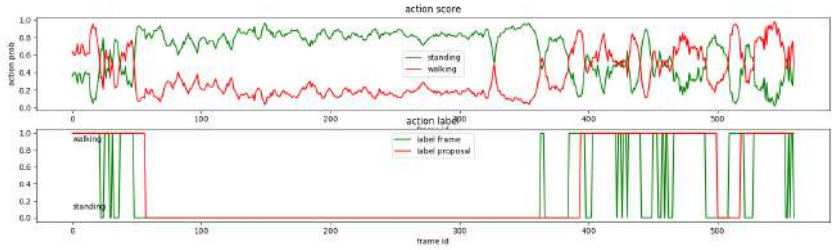
At the stage of training, we utilize the PoseNet model as described here in Figure 4.39 and generate pose tensor of 10 frames. Here we have trained two models, one is trained exactly the way the pipeline works, which is generating pose tensor at each frame (expect for the first 10 frames), the other is trained with data jittering augmentation, which is supposed to compensate the negative effects brought by the rapid change of camera perspective and instability of bounding box obtained from *Person Tracker*. Figure 5.6 presents the action proposal results with jitter model and stride-1 model.

Implementation

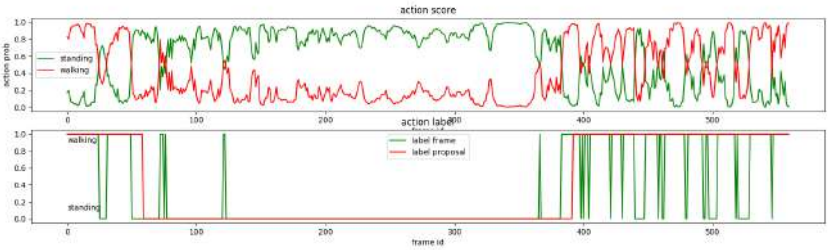
- **Experiments 1: Ground Truth (offline)**

  The first experiment we conduct after training PoseNet model is to apply them on the HDF5 files to validate the effectiveness of data processing together with the action grouping strategy in the pipeline. What we do is to read the pose keypoints from the HDF5 files and store them in numpy arrays (scripts written in Python). Then we generate pose tensor the way the pipeline works. At last we write the final action labels to file and group the same action category with the corresponding start and end frame IDs. Since in our baseline, we have the start and end frames of an action performed by a single pedestrian in one video clip, so we arrange the action classification results into the same pattern, i.e. [start_frame, end_frame, action]. After that, we take the Intersection over Union[4] (IoU) between our experiment results and the baseline as the evaluation metric, which is illustrated in Figure 5.7.

[3] `https://en.wikipedia.org/wiki/Hierarchical_Data_Format`
[4] `https://en.wikipedia.org/wiki/Jaccard_index`



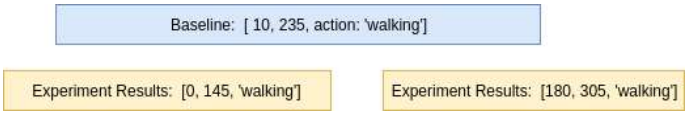
(a) Action proposal plotting with jitter model



(b) Action proposal plotting with stride-1 model

Figure 5.6: A comparion of action proposal results between jitter model and stride-1 model. The action scores plotted in (a) appear more stable than the results with stride-1 model in (b)

(a) Example case 1



(b) Example case 2

Figure 5.7: Example cases of IoU calculation between baseline and expriment results. In case 1, the Union of two results is the frame range [0, 235], and the Intersection is [10, 225], then the IoU = Intersection / Union, which is (225 - 10 + 1) / (235 - 0 + 1) = 91.5%. While in case 2, the Union is [0, 305], and the Intersection is [10, 145] ∪ [180, 235], so the IoU is ((145 - 10 + 1) + (235 - 180 + 1)) / (305 - 0 + 1) = 62.7%

- **Experiment 2: Ground Truth (online)**

  The second experiments we plan is to feed the image sequences as input to the pipeline, trying to verify that the detection, tracking, pose estimation, data processing as well as action classification (including action grouping) can coordinate with each other reasonably. We also make the comparion of the results between this experiment and the baseline just like the offline testing occasion.

- **Experiment 3: Original Videos from JAAD**

  After running testing on ground truth, we now run the pipeline on the original videos in the JAAD dataset. Unlike the ground truth, the original videos contains many

more agents, a quantitive comparison between the runtime results and baseline is impossible. One possible scenario is shown in Figure 5.8. At the end the action classification results of runtime testing on video clips will be demonstrated in the form of action-label-rendered videos, and the runtime performance (fps) of each node in the pipeline will be evaluated both on Nvidia Jetson TX2 and GPU Server.

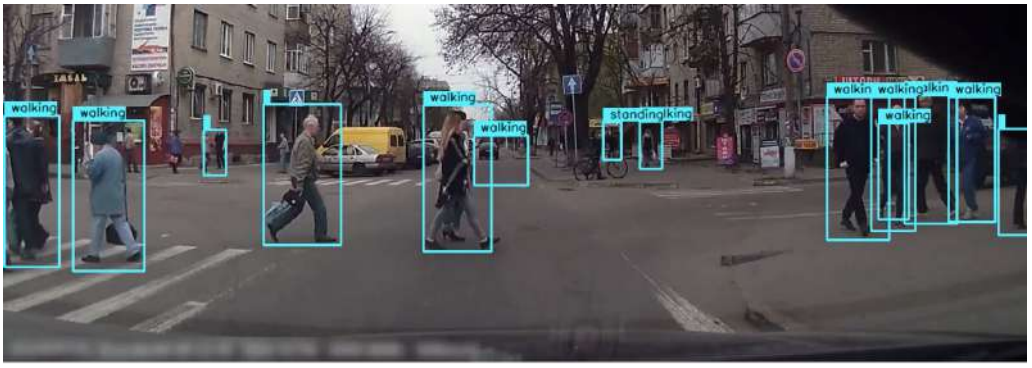


Figure 5.8: A crowded scenario in the JAAD dataset.

5.4 Results and Discussion

In this section we will present the results of the experiments, and evaluate the performance on the metrics of IoU (%) and frame rate (fps) of ROS nodes. We have also roughly measured the latency between nodes based on the time stamp in the measage Header[5], which ranges from about 1 ms to 6 ms depending on the message size. The measurement of message latency is conducted on Nividia Jetson TX2.

Results

In the experiments, we set two main variables: PoseNet model, testing fashion. We also suppose that in experiment 2, only the pedestrians with an IoU of at least 0.5 (threshold) are successfully proceeded, as our pipeline only classifies two categories (walking or standing), theoretically the probability of either action in with random prediction would be 0.5. Table 5.1 shows the results of experiment 1 and 2.

Here the two groups of ground truth are evaluated with the PoseNet model with and without jittering augmentation respectively. In the column of **IoU**, two IoU evaluation thresholds are set to 0 and 0.5 repectively. Apart from the value of IoU, the number of total videos and that of valid videos with an IoU $\geq$ threshold are also listed below.

Another evaluation is illustrated in Table 5.2 and 5.3, which present the runtime performance in fps both on Nvidia Jetson TX2 and GPU Server in multi-person and single-person scenarios respectively. The testing is running on the JAAD videos, and the frame rate of each node in the pipeline is evaluated.

Discussion

As we can see from Table 5.1, the offline ground truth testing results in a very high alignment with baseline, reaching the value almost the same as the accuracy of PoseNet jittering model and stride-1 model (both 0.8596). With an IoU threshold of 0.5, the IoU of the total valid videos could reach over 90% of the baseline. Here jittering model doesn't

[5] `http://docs.ros.org/melodic/api/std_msgs/html/msg/Header.html`

Table 5.1: Experiment results: ground truth offline and online testing

	IoU	
	Threshold = 0	Threshold = 0.5
Ground Truth (offline) jittering model (accuracy: 0.8596)	0.878 Total videos: 698 valid videos: 662	0.902 Total videos: 698 valid videos: 636
Ground Truth (offline) stride-1 model (accuracy: 0.8596)	0.896 Total videos: 698 valid videos: 668	0.921 Total videos: 698 valid videos: 640
Ground Truth (online) jittering model (accuracy: 0.8596)	0.543 Total videos: 676 valid videos: 588	0.716 Total videos: 676 valid videos: 337
Ground Truth (online) stride-1 model (accuracy: 0.8596)	0.519 Total videos: 442 valid videos: 397	0.708 Total videos: 442 valid videos: 218

Table 5.2: Run-time performance of ROS node in muti-person scenario

ROS node	**FPS**
/person_detector	Nvidia Jetson TX2: 2 ∼3 GPU Server: 35 ∼40
/person_tracker	Nvidia Jetson TX2: 3 ∼5 GPU Server: 4 ∼5
/pose_3d_ros	Nvidia Jetson TX2: 3 ∼5 GPU Server: 10 ∼15
/openpose_ros	Nvidia Jetson TX2: 0.8 ∼1.2 GPU Server: 8 ∼10
/data_manager_3d	Nvidia Jetson TX2: 30 ∼40 GPU Server: ∼100
/pose_net_3d	Nvidia Jetson TX2: 15 ∼20 GPU Server: 40 ∼50
/visual_agency	Nvidia Jetson TX2: 20 ∼30 GPU Server: ∼50

have a better performance than stride-1 model, as offline pose data dosen't change on the running, the jittering effect doesn't appear obvious. Based on the results of offline testing, we could verify that the data processing for pose tensor generation is also valid during run-time process.

However, for the online testing on ground truth, the IoU shows an obvious lower value, and almost half of the videos are invalid with the IoU threshold of 0.5. In the case of 0 as IoU threshold, the final IoU of all valid videos gets worse, reaching at around 50%. On

Table 5.3: Run-time performance of ROS node in single-person scenario

ROS node	FPS
/person_detector	GPU Server: 35 ∼ 40
/person_tracker	GPU Server: 50 ∼ 80
/pose_3d_ros	GPU Server: 15 ∼ 25
/openpose_ros	GPU Server: 10 ∼ 15
/data_manager_3d	GPU Server: ∼ 250
/pose_net_3d	GPU Server: ∼ 50
/visual_agency	GPU Server: ∼ 50

the other hand, however, the jittering model as expected presents a better performance on run-time occasion.

Some possible explanations are here to be discussed.

- **Detection and tracking failure**

  As mentioned in Section 4.2.3 and 4.2.4, the person detector and person tracker do not fit in all scenarios with the same parameters settings. After some exploration, we find that the positioning of the person bounding box plays a key role here. As in our pipeline, the bounding box obtained from tracking is then utilized to crop the person image from the image frame, and then it will be resized and zero-padded to a 256 × 256 patch as shown in Figure 4.28, before the pose estimator extracts the pose keypoints. As a result, the person bounding box should be cropped carefully, or the occasion illustrated in Figure 5.9 will probably happen.

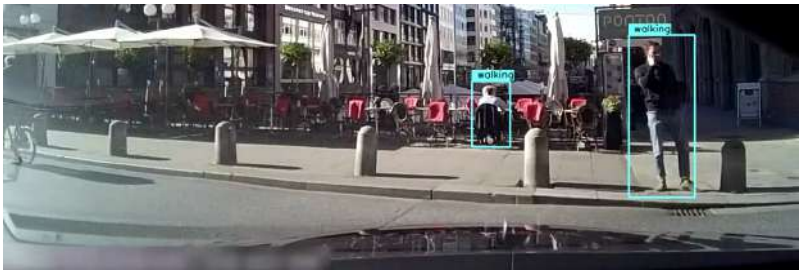
  Another problem that occurs in *Person Tracker* is the multiple initialization of one single pedestrian's track. As shown in Figure 5.10, the tracking of the same pedestrian is initialized multiple times. As a result, when running the testing, we will get multiple persons' action prediction, but in fact, they are the same person.

(a) (b) (c)

Figure 5.10: An example scenario of multiple person IDs from JAAD Ground Truth.

Apart from the mentioned two problems, Figure 5.11 and Figure 5.12 also present

(a) Camera perspective 1



(b) Camera perspective 2

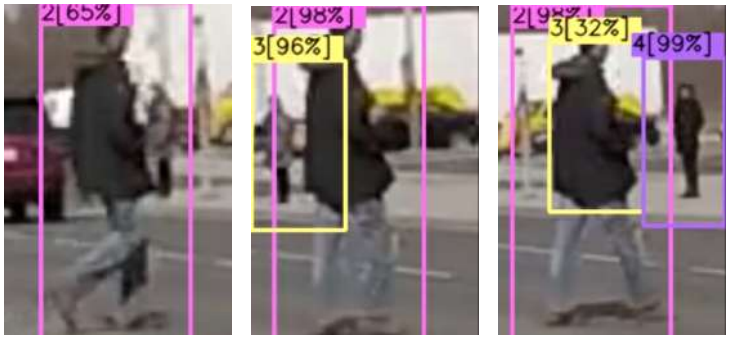
(c) Camera perspective 3

Figure 5.9: An example scenario of camera perspective shifting and person bounding box floating. A negitive effect could occur when the perspective of the camera changes rapidly, which often happens at the border area of the camera field. As shown from (a) to (c), the bounding box of the person would also change both in size and position, thus leading to a false joint positions after the padding of cropped person image. Even the person is standing, the shifting of the joints between consecutive frames will be predicted as the person is moving. Thus more robust PoseNet models are expected to be trained with more contextual information.

multi-person bounding box intersection and occlusion occasions that also make it more difficult for a clean tracking.

- **Action grouping strategy**

  The second part that might affect the effectiveness of the pipeline is the online action grouping. As is described in the algorithm 1, we have two hyperparameters, which are choosen empirically. As defaul we set τ, the action threshold, to 0.5, γ, the tolerance threshold, to 10 (frames). However, the strategy could bring some delay for action proposal. Figure 5.13 shows the online action scores and actions labels plotting. A reasonable action proposal should be walking along all the frames, because the walking action scores apparently surpass that of standing, while after action grouping by our strategy, the action isn't proposed as walking until about the 20th frame.



(a) (b) (c)

Figure 5.11: An example scenario of multiple-person bounding box intersection from JAAD Ground Truth.

(a) (b) (c)

Figure 5.12: An example scenario of agent occluded by others from JAAD Ground Truth.

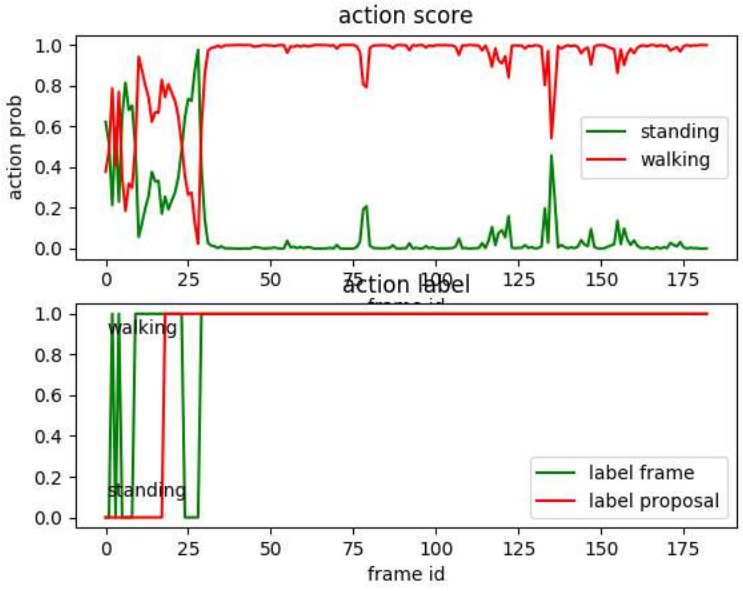
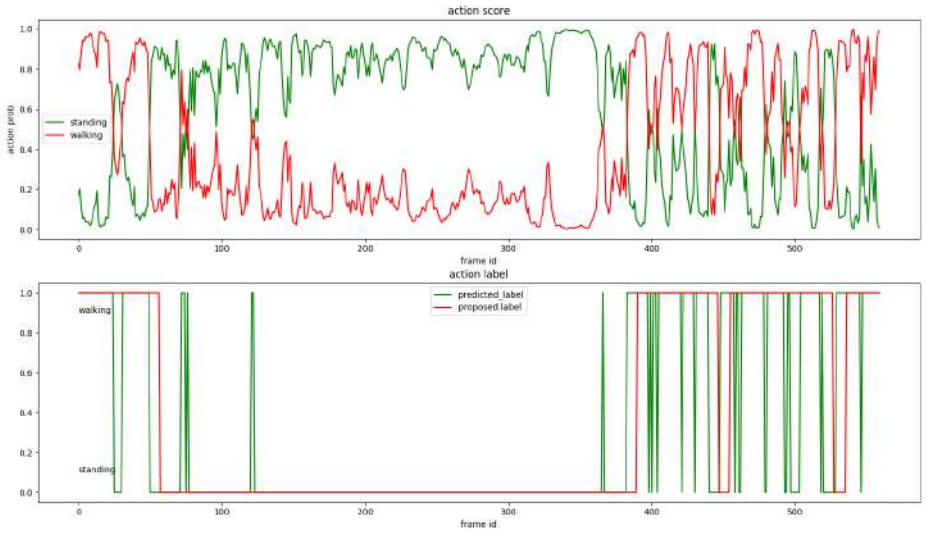


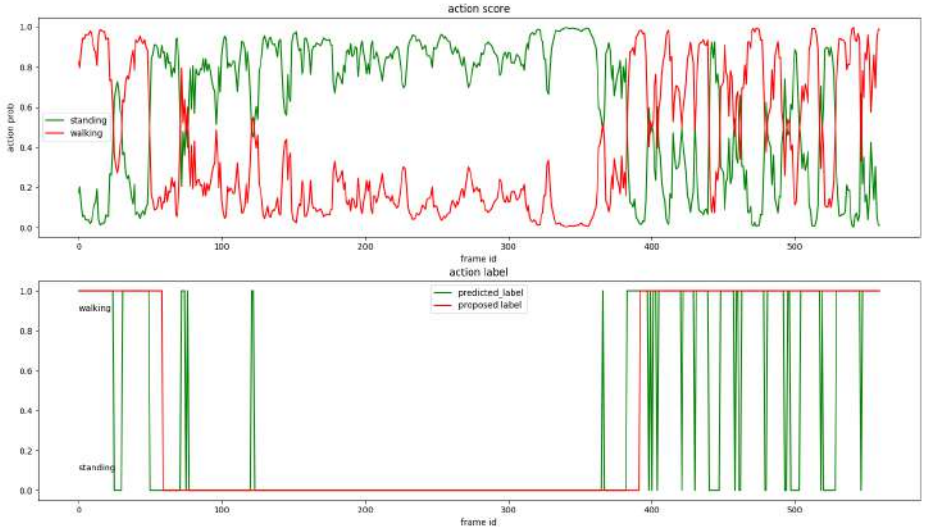
Figure 5.13: An example of the delay for action proposal.

Figure 5.14 displays the action proposal results with two different tolerance thresholds. As we can see, smaller tolerance threshold responds to the action changing

more quickly, while larger tolerance threshold allows more noises and get a smoothier action proposal result.



(a) Action proposal with $\gamma = 8$



(b) Action proposal with $\gamma = 10$

Figure 5.14: A comparison between two diffent tolerance thresholds.

6. Conclusion and Future Work

In the last chapter, we will present conclusions that we have drawn from the experiment results. Then some limitations in the proposed pipeline will be discussed and potential future works will be introduced.

6.1 Conclusion

In this thesis, we described the architecture design of a pipeline for human action detection and recognition based on the ROS framework. We focused on how ROS tools and functionalities led to a modular code structure, easy to configure and proposed seven modules, i.e. *Source Provider*, *Person Detector*, *Person Tracker*, *Feature Extractor*, *Data Manager*, *Action Predictor* and *Visual Agency* to effectively perform human action detection and recognition from video streams and image sequences. In each module, we introduced its functionality and explained related algorithms and the frameworks we referred to, followed by some implementation details with the APIs in the module. We also proposed an online action proposal strategy, which reduces a lot of noise in action prediction and groups the actions of the same category reasonably. Experiments were performed on JAAD dataset to evaluate the effectiveness of the pipeline with the metric of IoU between the experiment reults and ground truth. The run-time performance was evaluated with the metric of frame rate both on Nvidia Jetson TX2 and GPU Server. Experiment results proved the effectiveness of the online data processing (interpolation, normalization, pose tensor, etc.) and semi-real-time performance of our pipeline with proper GPU support, leaving the bottle neck in person tracking module.

6.2 Future Works

As future works, we firstly envision to improve the robustness and run-time performance of *Person Tracker* module. The current person tracker framework follows "tracking-by-detection" paradigm, in the future we would propose a framework which combines detection and tracking into one module with image frames as input and person IDs as output. Moreover, we are planning to merge the branches of 2D pose and 3D pose in the pipeline, even further, to integrate other features into the *Feature Extractor* module to form a more sophisticated and robust feature pool for action recognition. Besides, in terms of ROS framework, we propose to transfer the pipeline from ROS1 to ROS2, which has introduced many new features and shares an improved real-time performance.

List of Figures

List of Tables

Bibliography

[AHP06] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037–2041, Dec 2006.

[Ard17] J. S. H. K. S. L. M.-N. F. A. G. Arduengo, M., "Ros wrapper for real-time multi-person pose estimation with a single camera," Tech. Rep., 2017.

[ARS06] A. Adam, E. Rivlin, and I. Shimshoni, "Robust fragments-based tracking using the integral histogram," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, June 2006, pp. 798–805.

[Avi07] S. Avidan, "Ensemble tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 2, pp. 261–271, Feb 2007.

[BAT11] W. Brendel, M. Amer, and S. Todorovic, "Multiobject tracking as maximum weight independent set," in *CVPR 2011*, June 2011, pp. 1273–1280.

[BC86] T. J. Broida and R. Chellappa, "Estimation of object motion parameters from noisy images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 1, pp. 90–99, Jan 1986.

[BETG08] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346 – 359, 2008, similarity Matching in Computer Vision and Multimedia. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314207001555

[Bla04] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 5–18, Jan 2004.

[BM11] T. Brox and J. Malik, "Large displacement optical flow: descriptor matching in variational motion estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 3, pp. 500–513, 2011. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2011/Bro11a

[BR11] B. Benfold and I. Reid, "Stable multi-target tracking in real-time surveillance video," in *CVPR 2011*, June 2011, pp. 3457–3464.

[BWM17] F. Baradel, C. Wolf, and J. Mille, "Pose-conditioned spatio-temporal attention for human action recognition," *CoRR*, vol. abs/1703.10106, 2017. [Online]. Available: http://arxiv.org/abs/1703.10106

[BYB09] B. Babenko, M. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 983–990.

[CLL05] R. T. Collins, Y. Liu, and M. Leordeanu, "Online selection of discriminative tracking features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1631–1643, Oct 2005.

[CRM03] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, May 2003.

[CSWS17] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *CVPR*, 2017.

[CV95] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995. [Online]. Available: https://doi.org/10.1023/A:1022627411411

[Dat18] A. Dattalo. (2018) Ros introduction. Online Wiki. [Online]. Available: http://wiki.ros.org/ROS/Introduction

[DHG+14] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *CoRR*, vol. abs/1411.4389, 2014. [Online]. Available: http://arxiv.org/abs/1411.4389

[DT05] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, June 2005, pp. 886–893 vol. 1.

[DTS06] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," vol. 3952, 05 2006, pp. 428–441.

[DVM11] T. B. Dinh, N. Vo, and G. Medioni, "Context tracker: Exploring supporters and distracters in unconstrained environments," in *CVPR 2011*, June 2011, pp. 1177–1184.

[EHD00] A. M. Elgammal, D. Harwood, and L. S. Davis, "Non-parametric model for background subtraction," in *Proceedings of the 6th European Conference on Computer Vision-Part II*, ser. ECCV '00. London, UK, UK: Springer-Verlag, 2000, pp. 751–767. [Online]. Available: http://dl.acm.org/citation.cfm?id=645314.649432

[FB81] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[FGMR10] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, Sept 2010.

[FPZ16] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," *CoRR*, vol. abs/1604.06573, 2016. [Online]. Available: http://arxiv.org/abs/1604.06573

[FT07] J. Ferryman and D. Tweed, "An overview of the pets 2007 dataset." in *International Workshop on Performance Evaluation of Tracking and Surveillance (PETS)*, 2007.

[GDDM13] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[GGB06] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting," in *BMVC*, 2006.

[Gir15] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[HGS+16] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M. Cheng, S. L. Hicks, and P. H. S. Torr, "Struck: Structured output tracking with kernels," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 2096–2109, Oct 2016.

[HS81] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *ARTIFICAL INTELLIGENCE*, vol. 17, pp. 185–203, 1981.

[HS97] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[IS] Y. . . T. . R. u. . h. Isaac Saito, Howpublished = Online Wiki.

[JGZ+13] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black, "Towards understanding action recognition," in *2013 IEEE International Conference on Computer Vision*, Dec 2013, pp. 3192–3199.

[Kha18] M. U. Khalid, "Pose-based action recognition in video survelliance," 2018, master thesis, Technical University of Dortmund, Dortmund.

[KHN10] C.-H. Kuo, C. Huang, and R. Nevatia, "Multi-target tracking by on-line learned discriminative appearance models," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 685–692, 2010.

[KJG+11] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: A large video database for human motion recognition," in *2011 International Conference on Computer Vision*, Nov 2011, pp. 2556–2563.

[KMM10] Z. Kalal, J. Matas, and K. Mikolajczyk, "P-n learning: Bootstrapping binary classifiers by structural constraints," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010, pp. 49–56.

[Kou18] A. Koubaa. (2018) Ros services. Online Wiki. [Online]. Available: http://wiki.ros.org/Services

[KRT16] I. Kotseruba, A. Rasouli, and J. K. Tsotsos, "Joint attention in autonomous driving (JAAD)," *CoRR*, vol. abs/1609.04741, 2016. [Online]. Available: http://arxiv.org/abs/1609.04741

[KSEH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 25, 01 2012.

[KST+06] N. Katsarakis, G. Souretis, F. Talantzis, A. Pnevmatikakis, and L. Polymenakos, "3d audiovisual person tracking using kalman filtering and information theory," in *CLEAR*, 2006.

[LAE+15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[Lin18] W. Lin, "Monocular 3d human pose-based action recognition," 2018, master thesis, Technical University of Munich, Munich.

[LK81] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *In IJCAI81*, 1981, pp. 674–679.

[LMB+14] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[LMSR08] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[Low04] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[LSXW16] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Spatio-temporal LSTM with trust gates for 3d human action recognition," *CoRR*, vol. abs/1607.07043, 2016. [Online]. Available: http://arxiv.org/abs/1607.07043

[LZS17] T. Lin, X. Zhao, and Z. Shou, "Single shot temporal action detection," *CoRR*, vol. abs/1710.06236, 2017. [Online]. Available: http://arxiv.org/abs/1710.06236

[MA10] J. Müller and M. Arens, "Human pose estimation with implicit shape models," in *Proceedings of the First ACM International Workshop on Analysis and Retrieval of Tracked Events and Motion in Imagery Streams*, ser. ARTEMIS '10. New York, NY, USA: ACM, 2010, pp. 9–14. [Online]. Available: http://doi.acm.org/10.1145/1877868.1877873

[MBM+13] M. Munaro, F. Basso, S. Michieletto, E. Pagello, and E. Menegatti, *A Software Architecture for RGB-D People Tracking Based on ROS Framework for a Mobile Robot.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 53–68. [Online]. Available: https://doi.org/10.1007/978-3-642-35485-4_5

[MPP01] A. Mohan, C. Papageorgiou, and T. Poggio, "Example-based object detection in images by components," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 4, pp. 349–361, Apr. 2001. [Online]. Available: https://doi.org/10.1109/34.917571

[NHV+15] J. Y. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," *CoRR*, vol. abs/1503.08909, 2015. [Online]. Available: http://arxiv.org/abs/1503.08909

[NYD16] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*, 2016, pp. 483–499. [Online]. Available: https://doi.org/10.1007/978-3-319-46484-8_29

[PF18] M. W. M. H. Péter Fankhauser, Dominic Jud. (2018) Programming for robotics - ros. University Lecture. [Online]. Available: http://www.rsl.ethz.ch/education-students/lectures/ros.html

[POP98] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan 1998, pp. 555–562.

[RBR] Y. . . T. . R. u. . h. Radu B. Rusu, Howpublished = Online Wiki.

[RDGF15] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[RF18] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[RHGS15] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[RKT17] A. Rasouli, I. Kotseruba, and J. K. Tsotsos, "Are they going to cross? a benchmark dataset and baseline for pedestrian crosswalk behavior," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 206–213.

[Rom14] A. Romero. (2014) Ros concepts. Online Wiki. [Online]. Available: http://wiki.ros.org/ROS/Concepts

[RS13] K. K. Reddy and M. Shah, "Recognizing 50 human action categories of web videos," *Mach. Vision Appl.*, vol. 24, no. 5, pp. 971–981, Jul. 2013. [Online]. Available: http://dx.doi.org/10.1007/s00138-012-0450-4

[SB03] H. Sidenbladh and M. J. Black, "Learning the statistics of people in images and video," *Int. J. Comput. Vision*, vol. 54, no. 1-3, pp. 181–207, Aug. 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1023765619733

[SG99] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 2, June 1999, pp. 246–252 Vol. 2.

[SJMS17] T. Simon, H. Joo, I. A. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," *CoRR*, vol. abs/1704.07809, 2017. [Online]. Available: http://arxiv.org/abs/1704.07809

[SLNW16] A. Shahroudy, J. Liu, T. Ng, and G. Wang, "NTU RGB+D: A large scale dataset for 3d human activity analysis," *CoRR*, vol. abs/1604.02808, 2016. [Online]. Available: http://arxiv.org/abs/1604.02808

[ST94] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 1994, pp. 593–600.

[SZ14a] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *CoRR*, vol. abs/1406.2199, 2014. [Online]. Available: http://arxiv.org/abs/1406.2199

[SZ14b] ——, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[SZS12] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," *CoRR*, vol. abs/1212.0402, 2012. [Online]. Available: http://arxiv.org/abs/1212.0402

[Tho13] D. Thomas. (2013) Ros parameter server. Online Wiki. [Online]. Available: http://wiki.ros.org/Services

[TKBM99] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practice of background maintenance." IEEE Computer Society Press, September 1999, pp. 255–261. [Online]. Available: https://www.microsoft.com/en-us/research/publication/wallflower-principles-and-practice-of-background-maintenance/

[VJ01] P. Viola and M. Jones, "Robust real-time face detection," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, July 2001, pp. 747–747.

[VJS03] Viola, Jones, and Snow, "Detecting pedestrians using patterns of motion and appearance," in *Proceedings Ninth IEEE International Conference on Computer Vision*, Oct 2003, pp. 734–741 vol.2.

[VL15] A. Vedaldi and K. Lenc, "Matconvnet: Convolutional neural networks for matlab," in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM '15. New York, NY, USA: ACM, 2015, pp. 689–692. [Online]. Available: http://doi.acm.org/10.1145/2733373.2807412

[VVV15] S. Vasuhi, M. Vijayakumar, and V. Vaidehi, "Real time multiple human tracking using kalman filter," in *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, March 2015, pp. 1–6.

[WB95] G. Welch and G. Bishop, "An introduction to the kalman filter," Chapel Hill, NC, USA, Tech. Rep., 1995.

[WHY09] X. Wang, T. X. Han, and S. Yan, "An hog-lbp human detector with partial occlusion handling," in *2009 IEEE 12th International Conference on Computer Vision*, Sept 2009, pp. 32–39.

[WKSL11] H. Wang, A. Kläser, C. Schmid, and C. Liu, "Action recognition by dense trajectories," in *CVPR 2011*, June 2011, pp. 3169–3176.

[WRKS16] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines." *CoRR*, vol. abs/1602.00134, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1602.html#WeiRKS16

[WS13] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *2013 IEEE International Conference on Computer Vision*, Dec 2013, pp. 3551–3558.

[Wu18] Y. Wu. (2018) Ros master. Online Wiki. [Online]. Available: http://wiki.ros.org/Master

[WXW+16a] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool, "Temporal segment networks: Towards good practices for deep action recognition," *CoRR*, vol. abs/1608.00859, 2016. [Online]. Available: http://arxiv.org/abs/1608.00859

[WXW+16b] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Val Gool, "Temporal segment networks: Towards good practices for deep action recognition," in *ECCV*, 2016.

[XZW+17] Y. Xiong, Y. Zhao, L. Wang, D. Lin, and X. Tang, "A pursuit of temporal accuracy in general activity detection," *CoRR*, vol. abs/1703.02716, 2017. [Online]. Available: http://arxiv.org/abs/1703.02716

[ZCW+16] Y. Zhang, L. Cheng, J. Wu, J. Cai, M. N. Do, and J. Lu, "Action recognition in still images with minimum annotation efforts," *IEEE Transactions on Image Processing*, vol. 25, no. 11, pp. 5479–5490, Nov 2016.

[ZHS+17] X. Zhou, Q. Huang, X. Sun, X. Xue, and Y. Wei, "Weakly-supervised transfer for 3d human pose estimation in the wild," *CoRR*, vol. abs/1704.02447, 2017. [Online]. Available: http://arxiv.org/abs/1704.02447

[ZHZ08] C. Zhang, R. Hamid, and Z. Zhang, "Taylor expansion based classifier adaptation: Application to person detection," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[ZJZ10] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *Int. J. Machine Learning & Cybernetics*, vol. 1, pp. 43–52, 2010.

[ZZXW18] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," 2018, cite arxiv:1807.05511. [Online]. Available: http://arxiv.org/abs/1807.05511