Human Detection and Tracking

ENPM808X - Midterm Proposal

Nitesh Jha
Master of Engineering in Robotics
University of Maryland - College Park
niteshj@umd.edu

Tanuj Thakkar
Master of Engineering in Robotics
University of Maryland - College Park
tanuj@umd.edu

I. Software Plan

A. Overview

Acme Robotics is launching a multi-service human assistance robot next year. The operation of this robot relies on knowing precise locations of people in the vicinity. It utilizes this information to avoid collisions with humans, navigate to them to provide service, and to follow humans to assist in carrying heavy loads. Robot motion in a dynamic environment requires that the detection of obstacles is real-time in order for the robot to have sufficient time to actively avoid it. For this purpose, we are developing a module which detects humans and tracks their motion over time. There are many methods to detect humans. Some are based on extracting predetermined low-level features from the image, whereas some are based on learning relevant features which help to distinguish between a human and non-human. With the exponential growth of processing capabilites and the availability of large amount of data, data-driven models, such as CNNs are very popular and have shown very accurate results [3]. We aim to utilize a state-of-the-art object detection model, YOLO, which performs object detection and localization in a single pass [2]. This gives the model high FPS and is suitable for real time applications. In addition to detection, it is important to identify the same object/human in subsequent frames and keep track of it. This is useful not only for the execution of the service (to continue following the same human, approach a 'target' human) but also helps in cases where the detection module does not give detections continuously. For tracking, we plan to use OpenCV Tracking API such as KCF, MOSSE (based on correlation filters) [1].

B. Deliverables

- A C++ module to detect and track humans with a single RGB camera
- UML Diagrams
- Developer Documentation of the C++ module with ReadMe
- Code-base following Google styling guidelines for C++
- Software profiling using Valgrind
- TravisCI for continues integration and Coveralls for code coverage

C. Definitions and Acronyms

- YOLO: You Only Look Once
- CNN: Convolutional Neural Network
- KCF: Kernelized Correlation Filter
- MOSSE: Minimum Output Sum of Squared Error
- FPS: Frames Per Second

D. Methodology

The software module under consideration performs two main objectives, i.e., human detection and tracking. Hence, the module can be divided into three components, detection, tracking and transformation.

1) Detection: The detection component of the module will employ YOLO, You Only Look Once approach for human detection. It is a deep learning based approach using a CNN for single stage detection [2]. In contrast to other deep learning based approaches, as its name suggests, YOLO only looks at the entire image only once. It divides the image into smaller grids. Each grid is responsible to form N bounding boxes. Then, to avoid overlapping bounding boxes of the same class, it uses Intersection over Union as a measure to eliminate these and generate the final list of detections found in the image, along with the probability scores.

2) Tracking: After detecting humans, tracking them continuously is important in order to introduce reliability in the overall system when the detector fails. Continuing with the deep learning based approach, after an initial set of detections have been made with their respective bounding boxes, the tracking component will introduce unique identification for each of the detections. These unique identification is then used along with the bounding boxes to track the humans. The tracking module will be based on a multiple object tracking algorithm [1].

3) Transformation: Once the detections are obtained in the image coordinates, for the robot to be able to use them, we have to transform these coordinates to the robot reference frame, and also the world frame. Every pixel in the image corresponds to a 3D ray from the camera center to the object. Then, using an assumed height of human, the coordinates of the human is calculated. This is followed by a homogenous transformation to the robot reference frame. This information can then be used by the robot for providing the service.

II. Project Organization

A. Design

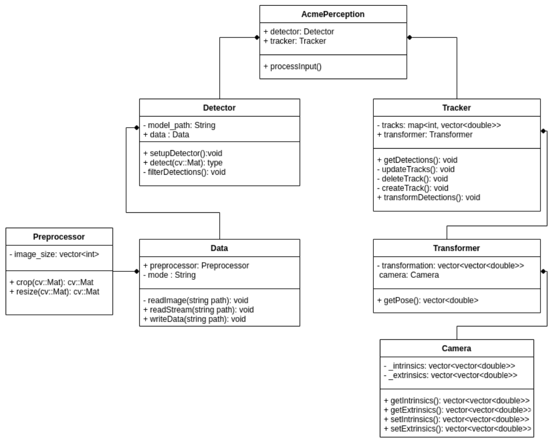
AcmePerception
+ detector: Detector
+ tracker: Tracker
+ processInput()
Detector
- model_path: String
+ data : Data
+ setupDetector():void
+ detect(cv::Mat): type
- filterDetections(): void
Tracker
- tracks: map<int, vector<double>>
+ transformer: Transformer
+ getDetections(): void
- updateTracks(): void
- deleteTrack(): void
- createTrack(): void
+ transformDetections(): void
Preprocessor
- image_size: vector<int>
+ crop(cv::Mat): cv::Mat
+ resize(cv::Mat): cv::Mat
Data
+ preprocessor: Preprocessor
- mode : String
- readImage(string path): void
+ readStream(string path): void
+ writeData(string path): void
Transformer
- transformation: vector<vector<double>>
camera: Camera
+ getPose(): vector<double>
Camera
- _intrinsics: vector<vector<double>>
- _extrinsics: vector<vector<double>>
+ getIntrinsics(): vector<vector<double>>
+ getExtrinsics(): vector<vector<double>>
+ setIntrinsics(): vector<vector<double>>
+ setExtrinsics(): vector<vector<double>>

Fig. 1: UML Class Diagram

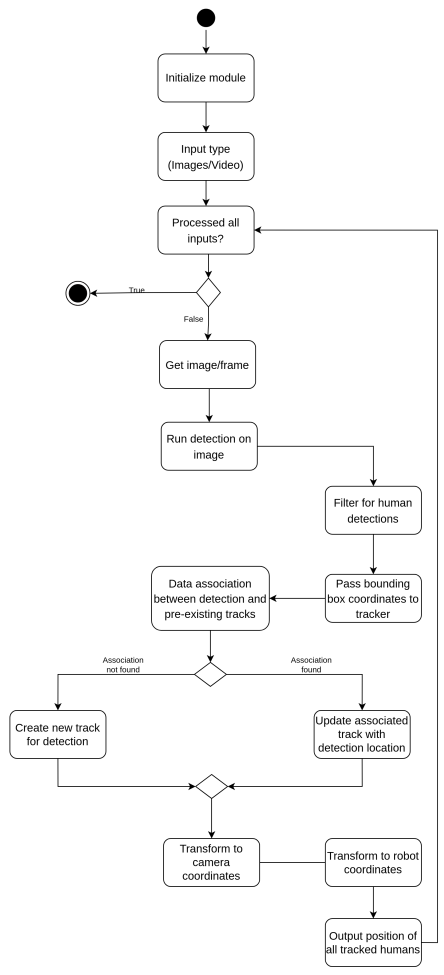
Initialize module
Input type (Images/Video)
Processed all inputs?
True
False
Get image/frame
Run detection on image
Filter for human detections
Pass bounding box coordinates to tracker
Data association between detection and pre-existing tracks
Association not found
Association found
Create new track for detection
Update associated track with detection location
Transform to camera coordinates
Transform to robot coordinates
Output position of all tracked humans

Fig. 2: UML Activity Diagram

B. Organizational structure

1) Phase 1
   - Nitesh Jha (Driver)
   - Tanuj Thakkar (Navigator)
2) Phase 2
   - Nitesh Jha (Navigator)
   - Tanuj Thakkar (Driver)

III. Managerial Process

A. Assumptions, dependencies, and constraints

- The YOLO detection model is pretrained for the purposes of this project
- There is no abrupt change in positions of humans we track
- We assume a human to be about 5 feet tall for transformation purpose.
- There are no significant occlusions between multiple tracked humans.
- Tracked humans going in and out of frame do not maintain same ID.

B. Monitoring and interfacing mechanisms

Git will be used for version control of the module code-base. For continuous integration, TravisCI will be employed along with Coveralls for tracking and analysing code coverage.

IV. Technical Process

A. Tools and Techniques

- Ubuntu 20.04 (LTS)
- C++ 14 or higher
- CMake 3.10.0 or higher
- Git
- TravisCI
- Coveralls
- Valgrind

B. Dependencies

- OpenCV 4.2.0
- Eigen 3.4.0
- Boost 1.71.0
- GTest
- GMock

C. Software Documentation

The documentation of the module at a developer level will be provided using Doxygen.

References

[1] Object tracking in computer vision. https://viso.ai/deep-learning/object-tracking/.

[2] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[3] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018.