EU Horizon 2020 Research & Innovation Program
Advanced Robot Capabilities & Take-Up
ICT-25-2016-2017
Grant Agreement No. 779942

Safe Robot Navigation in Dense Crowds

http://crowdbot.eu/project/

Technical Report

D 2.2: Local Sensing 1st Prototype

Work Package 2 (WP 2)
Local Sensing

Task Lead: RWTH
WP Lead: RWTH

Table of Contents

Abstract

In this report we present our first prototype of the perception pipeline developed for the CROWDBOT project. Currently, the focus of the perception pipeline is on detecting and tracking pedestrians in low to medium density scenarios using RGB-D cameras and 2D LiDAR sensors. We begin with reviewing the major detection and tracking methods used in our perception pipeline, as well as their ROS implementation. Then we proceed with quantitative evaluations, presenting results on the detection, tracking, and run-time performance. Finally, we discuss some on-going work to extend the perception pipeline, including interactive data annotation tools, optical flow aided pedestrian tracking, and detailed person analysis modules.

1. Introduction

Robot navigation within populated environments such as airports or train stations is a core component of the CROWDBOT project. For carrying out such navigation (and all the other following actions), advanced perception capabilities are a prerequisite, in particular the capability to sense pedestrians. The ideal perception strategy varies with respect to the crowd density of the environment. At low to medium crowd densities, where most pedestrian instances are visible to the robot and distinguishable in sensor readings, it is desirable to detect, to precisely localize, and to track each pedestrian instance. Pedestrian detection and tracking also provides the possibility for further analysis for specific pedestrian instances. On the other hand, when the surroundings of the robot become cluttered with a high density crowd (imagine the train station at rush hour), the individual pedestrian instances will no longer be distinguishable. In such a scenario, it may be more beneficial to rely on low-level perception cues, for example scene flow, to make navigation related decisions.

In this report we present our first prototype of perception pipeline. Our current perception pipeline focuses on tracking pedestrians in low to medium densities, where the pedestrian instances can be reliably detected from sensor readings (image, laser scan).

1.1 Overview of Perception Tracking Architecture

Our perception pipeline is designed following the well-known tracking-by-detection paradigm. Under this paradigm, objects are detected for each frame independently, and a tracking algorithm is used to associate detections that belong to the same object instance over multiple frames. We also adapt a modularized design approach. For each perception sensor, a detection module is instantiated that operates independently of the other sensors. The detections produced by the different modules are then transformed into a common 3D coordinate frame; detections from sensors with overlapping fields-of-view are fused; and the resulting fused 3D observations are fed to the tracking algorithm. The final output is a set of 3D object trajectories on the ground plane in either world coordinates (if robot odometry is provided) or in local 3D coordinates relative to the robot (if no odometry is available). The overall architecture is shown in Figure 1.

Our modularized design approach allows for easy adaptation to changes of sensor configuration. The only requirement for adapting to a new set of sensors or a different arrangement of the existing ones is an extrinsic calibration of each sensor relative to the robot coordinate system (as reflected in the ROS coordinate tree). Our current implementation provides dedicated person detection modules for RGB/RGB-D cameras and for 2D LiDAR sensors, which covers the two most important sensor types used in CROWDBOT. As a result, our tracking pipeline can be deployed for all three CROWDBOT robots (ETH's Augmented Pepper, UCL's Wheelchair, Locomotec's CUYBOT), even though they use different numbers of sensors in different configurations. **Figure 2** shows an example of our perception pipeline at work, using a recording collected from ETH's Augmented Pepper. The perception pipeline is implemented in ROS. An example rosgraph is shown in Figure 3.

In the following sections, we present in detail our detection module and tracking algorithm (Sections 2 and 3, respectively).

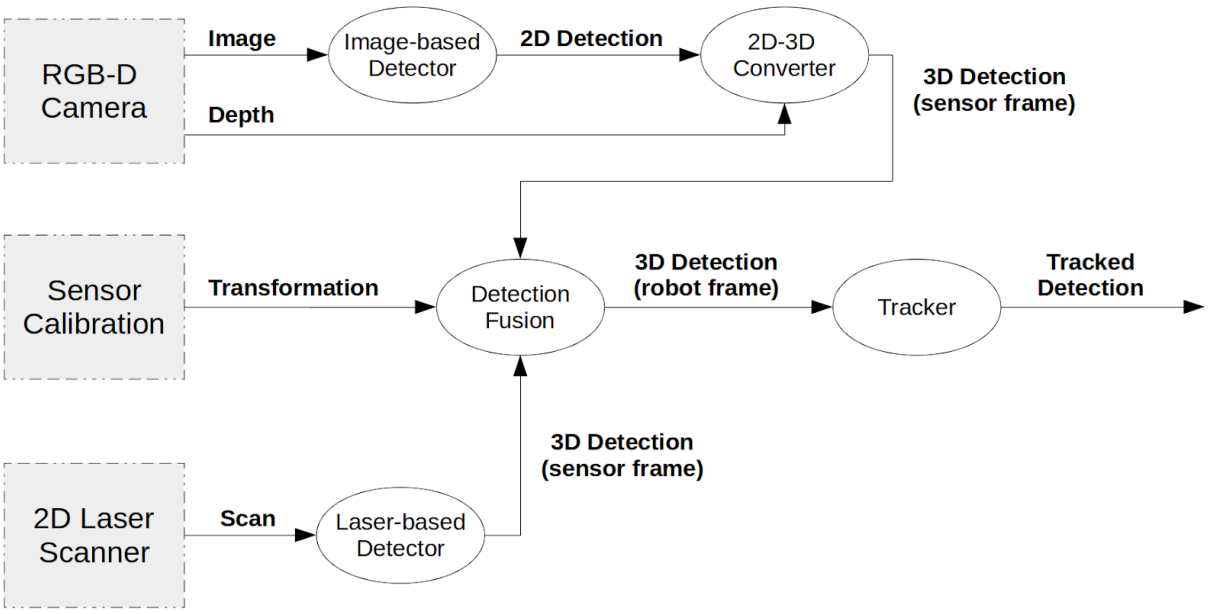


Figure 1. Architecture of our perception pipeline. We use a modularized design approach that enables a flexible combination of different numbers and arrangements of sensors. Dedicated detection modules detect pedestrians from each sensor independently. These detections are then transformed into a common coordinate frame; detections from sensors with overlapping viewing fields are fused; and the resulting measurements are passed to the 3D tracking algorithm.

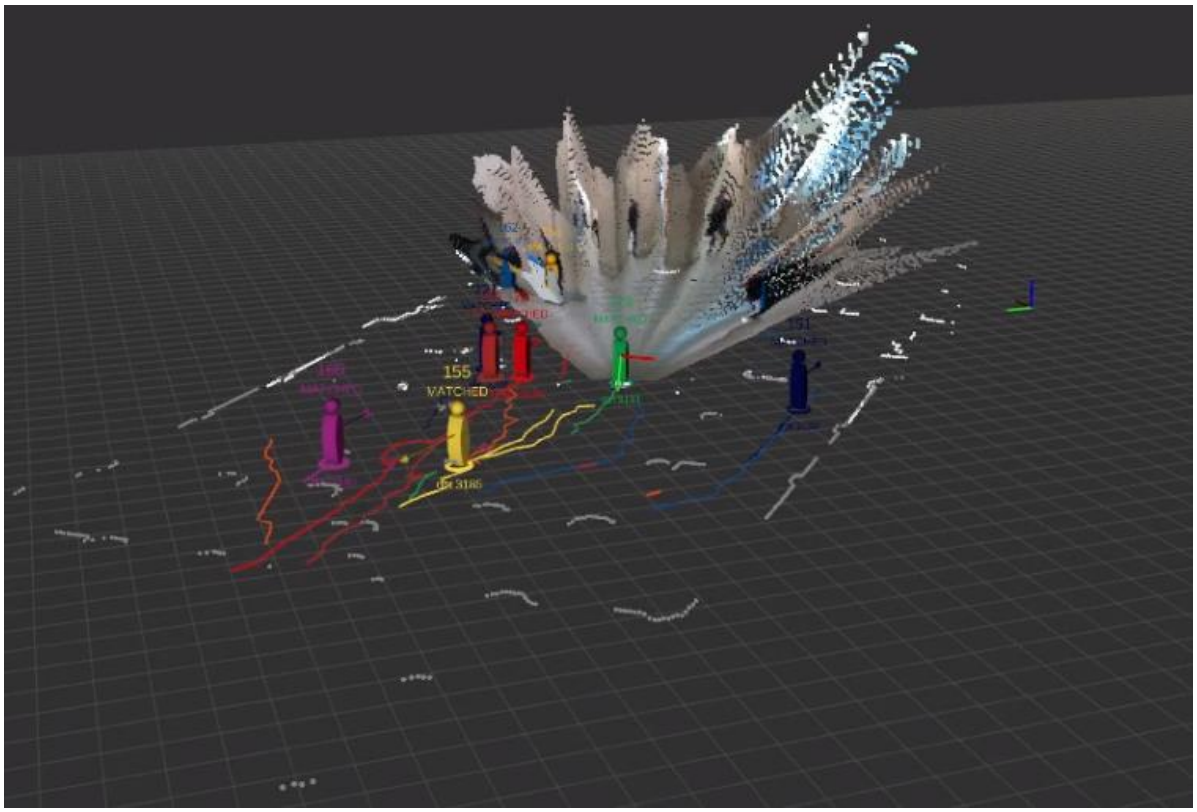


Figure 2. A screenshot of our perception pipeline running in ROS. The data is collected using ETH Augmented Pepper, which is equipped with two 2D laser scanners, viewing front (white points) and rear (gray points), as well as a front-facing RGB-D camera. Tracked pedestrians are shown as markers along with their past trajectories. Colors encode the ID of each tracked pedestrian.

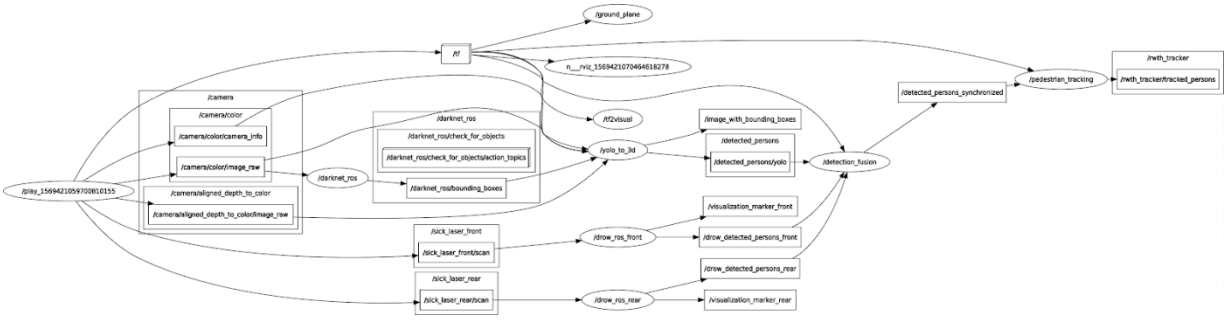


Figure 3. An example rosgraph of our perception pipeline.

2. Detectors

Detectors are the first components of our perception pipeline. In its most general form, an object detector aims to recognize the presence of objects belonging to some categories of interest, as well as to precisely localize the object instances from raw sensor measurements. Such detections often serve as input for further downstream analysis. In the CROWDBOT project, we are interested in detecting pedestrians from both camera and 2D LiDAR input. Our pipeline includes implementations of multiple successful pedestrian detection methods that have been developed over the years. Each of these implementations stands as a complete, individual component, and can be easily swapped in or out of the overall perception pipeline. The first two detectors (groundHOG, RGB-D Upper-body) are mainly included for historical reasons; whenever sufficient compute capabilities are available, the more recent YOLO-v3 detector performs significantly better and is always to be preferred.

In this section we briefly review the working mechanism of each detector. Readers are kindly referred to the original publications for more detailed information.

2.1 GroundHOG

An early example of image-based pedestrian detection is presented in the work of Dalal & Triggs [Dalal05]. This detector adopts a sliding-window approach using a Histograms of Oriented Gradients (HOG) feature representation and a linear Support Vector Machine (SVM) classifier. To detect pedestrians of different sizes and at different distances, a search over scales needs to be performed, making the approach computationally demanding. Sudowe & Leibe [Sudowe11] addressed this issue by constraining the search space using geometric constraints from the ground plane. Combined with an optimized GPU implementation, this improved detection method is referred to as groundHOG. An implementation of this method is included in our perception pipeline.

2.2 RGB-D Upper-body Detector

HOG-based methods are designed to detect pedestrians whose full bodies are visible, and they thus struggle in scenarios where the camera is mounted in a low position, or where pedestrians are in close proximity to the camera. Such scenarios, however, will be common for the targeted application of the CROWDBOT project. To address this problem, we include in our pipeline an implementation of the depth-based upper-body detector proposed by Jafari et al. [Jafari14]. This approach requires depth measurement and detects the upper body of potential pedestrians by matching the depth measurement with a learned template. It provides accurate detection when the pedestrians of interest are close to the camera at very low computational cost, running at more than video frame rate on a single CPU core.

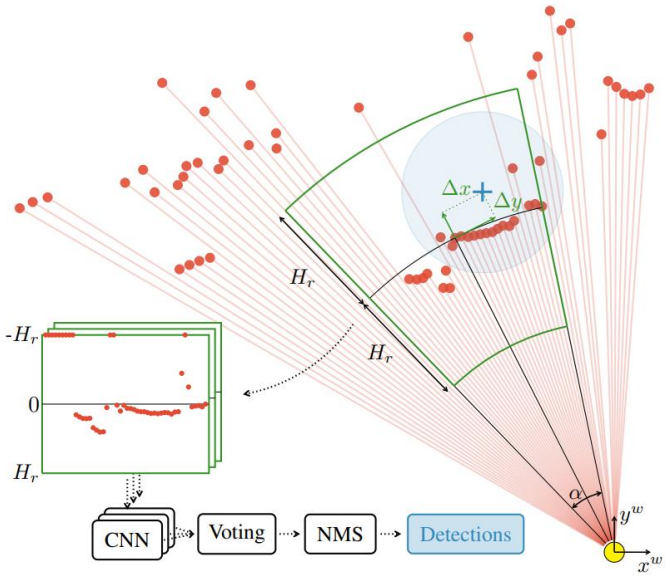


Figure 4. As a preprocessing step, the DROW detector cuts out a window for each laser point. The angular opening of the window is calculated from the distance measurement and the desired window width. Points within the window are then resampled to a fixed resolution, and passed into a detection network.

2.3 YOLO-v3

With the advent of deep learning methods, the Computer Vision and Robotics community has witnessed a significant increase in detector performance. One particular popular network for object detection is YOLO ("You Only Look Once") [Redmon16, Redmon18]. Designed as a single-stage detector with a customized light-weight backbone network, YOLO is able to detect objects in real-time, reaching 30+ fps on a desktop GPU. Thus, it is well-suited for robotic applications, and we use it as the main pedestrian detector in our pipeline. Over the years, YOLO has gone through several iterations, and we use the 3rd version, YOLOv3 [Redmon18].

To further reduce the onboard computation cost and power consumption, we have explored the possibility of running the YOLO detector with TensorRT optimizations. In Section XX, we present our run-time evaluation results on an NVidia Jetson AGX, a low-power embedded GPU designed for robotic applications. As our results show, YOLOv3 with TensorRT runs at 20+ fps on the Jetson AGX, making it a very attractive choice for the CROWDBOT platform.

2.4 2D Laser: DROW2 [ICRA/RA-L'18]

2D laser scanners are another important sensor type for many robotic platforms. Compared to an RGB-D camera, a 2D laser scanner provides more accurate depth measurements over a greater field-of-view and is more robust to lighting conditions. However, due to the absence of the rich appearance information from images, reliably detecting pedestrians from 2D laser scans is a challenging task. In their pioneering work, Beyer et al. proposed the first deep-learning based pedestrian detector from 2D laser scans [Beyer17, Beyer18]. Referred to as DROW (Distance RObust Wheelchair/Walker) detector, this approach is by far the most successful method to detect persons from 2D laser scans. In order to take advantage of the rich information from the 2D laser scanners on our robotic platform, we incorporate the DROW2 version of the detector [Beyer18] into our perception pipeline, with some modifications for the specific robotic platform, as described below.

The DROW detector consists of three steps: preprocessing, detection, and grouping (Figure 4). The preprocessing step cuts out a window around every laser point and resamples (interpolation or downsampling) the points within each window to a fixed resolution. Then a CNN is employed for each window of points to predict the classification label of the center point, as well as the offset of the predicted object center. Finally, the predicted object centers are grouped into instances with a voting mechanism. We refer the reader to the

original publication for more details ([Beyer18]). Note that the resampling mechanism employed in the preprocessing stage enables DROW to be used with laser scanning devices with different angular resolutions without requiring re-training -- a very convenient feature that we make extensive use of to accommodate the different sensors used within CROWDBOT.

We created a ROS node for the DROW detector to integrate with our perception pipeline. Some modifications were applied to achieve real-time performance. We trained the detector using the original dataset released by the authors, which were collected in an elderly care facility using a SICK S300 laser scanner mounted on a mobile robot at a height of 37cm above ground. The SICK S300 laser has a 225 degree field of view and 450 points per scan. The data are annotated with three class labels: wheelchairs, walkers, and pedestrians, as well as the object center location. The statistics of the dataset are summarized in **Figure 5**. We qualitatively tested the detector with data recorded in the hallway of the ETH main building, and have obtained very good results (Figure 6). The test data were collected by ETH using a Pepper platform with two SICK lasers (different model with the one used to collect training data) with 270 degree field of view and 811 points per scan, facing front and back, respectively. This result again confirms the DROW detector's capability to work with different 2D laser scanners without requiring retraining.

3. Tracker

While the detectors provide potential locations of pedestrians in a single sensor measurement, the tracker on the other hand, aims to connect detections that belong to the same pedestrian instance over time. A well-designed tracking algorithm could also alleviate the uncertainty in the detections, e.g., the false positives due to mis-classification or the false negatives due to occlusion, by evaluating the consistency between a single detection and its temporal correspondences in the track. In this section we present the tracking component of our pipeline. We begin with an overview of the theoretical foundations of the algorithm and proceed with the implementation details in the ROS environment.

3.1 Tracking Framework

We employ the minimum description length (MDL) tracking approach, first presented by Leibe et al. [Leibe08] and developed over time. The approach consists of two steps. The first step, known as hypothesis generation, builds an overcomplete set of track hypotheses from the past and current detections. The second step, multi-hypothesis handling, selects a subset of the generated hypotheses as valid tracks that explain the detections by solving an optimization problem.

A core component of the MDL tracker is a motion filter, which predicts the location of the tracked person in the current frame given its previous state (location and velocity), as well as updating its internal state based on supporting observations (detections). Here we use a Kalman filter with a constant velocity motion model.

3.2 Hypothesis Generation

In most other multi-object tracking approaches, such as MHT [Reid79], joint data association, i.e. the decision which detection to assign to which trajectory, usually comes first, and the velocity estimates are obtained as a result of this joint data association. In our experience, this works well as long as there are not too many interacting objects, but it causes a combinatorial explosion if many objects are observed close together. In contrast and following the framework developed by [Leibe08], our tracker operates the other way round. It starts by first generating a set of single-object trajectory hypotheses using independent Kalman filters and then selects a subset of those trajectory hypotheses that is mutually consistent. This means that trajectory generation is non-exclusive (i.e., multiple hypotheses may claim the same detection) and the velocity estimate can be obtained locally for each hypothesis just based on the associated detections. In our experience, this scales better to crowded scenes, since we have a more direct way to control priors on the trajectories (e.g., we could use knowledge about scene geometry in order to generate additional trajectory hypotheses aligned with expected crowd flow directions, which is not possible within MHT). For the later trajectory selection step, we solve a Quadratic Pseudo-Boolean Optimization (QPBO) problem using a scoring function to express the individual contribution of each trajectory and the pairwise interaction cost of each pair of overlapping trajectories.

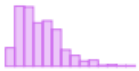
	Train	Validation	Test	Total
Sequences	78	30	5	113
Scans	341 138	74 744	48 131	464 013
Annotated Scans	17 665	3 919	2 428	24 012
Wheelchairs	14 455	5 595	1 970	22 020
Walkers	2 047	219	581	2 847
Wheelchairs by distance				
Walkers by distance				

Figure 5. Statistics of the data collected by Beyer et al. [Beyer18] in the original work of DROW detector. The DROW detector in our perception pipeline is trained using the same data.

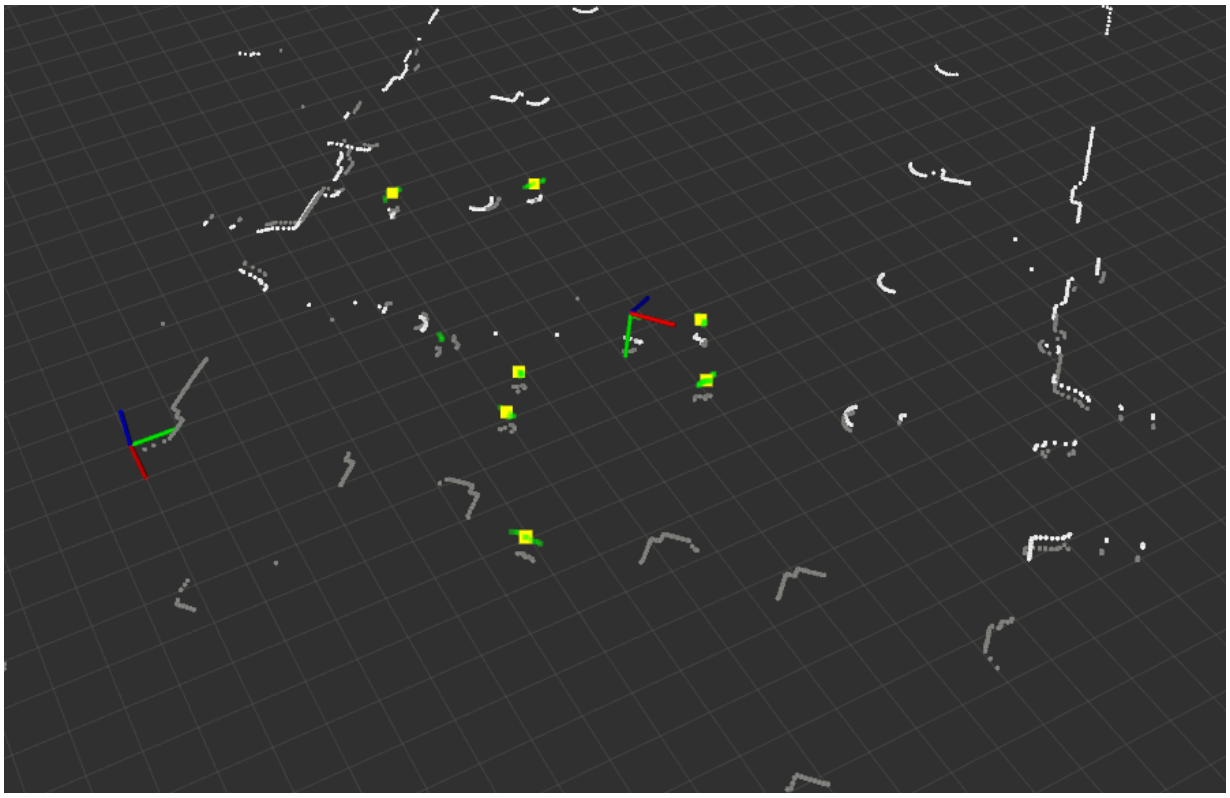


Figure 6. Results of the DROW detector on the ETH testing data. White and gray points denote the scan from the front and rear facing laser, respectively. Green markers are the predicted object centers from the points that have been classified as pedestrians. Yellow markers are centers of pedestrians after grouping.

Given a set of new detections at frame t, for each detection, the MDL tracker generates a new track. To improve the robustness of the new track, rather than simply assigning it with an initial velocity of zero, the MDL tracker first runs the Kalman filter backwards then forwards in time for a predefined time window. Then the resulting state (location and velocity) from the bi-directional Kalman filter pass is used as the starting state of the new track. Such an approach allows the newly generated track to incorporate detections from the past, and is done for every detection in the current frame.

The existing tracks from previous frame are simply propagated to the current frame using the Kalman filter. Combining the newly generated tracks and the existing tracks, we get an overcomplete set of track hypothesis. The next step is to select from this set the valid tracks that properly explain the scene observations (i.e., the detections).

3.3 Multi-Hypothesis Handling

To select valid tracks from the over complete set of hypothesis, the MDL tracker first assigns a score S_i to each track. The score of a track is based on the accordance between the predicted location from the motion model and the observed detection. Notice that a single detection can support multiple tracks. Optionally, appearance information from the image could also be taken into account for assigning a score to each track. Other than the single track score, the MDL tracker also assigns an interaction score S_ij between any two tracks. This score is modeled based on the overlapping parts between the two tracks as well as their supporting detections. It naturally includes the constraint that a single detection could not support two tracks at the same time.

With the single track score S_i and the interaction score S_ij, the task of selecting the best subset of tracks can be formulated as a quadratic pseudo-binary optimization problem (QPBO), which is solved using the technique presented by Schindler et al. [Schindler06]. Notice that while track generation is handled explicitly at the previous step, track termination is handled implicitly by hypothesis selection.

3.4 ID Assignment

One remaining task is to assign IDs to the selected tracks. If a track that has been selected is one that was extended from the previous frame, the same ID is assigned. On the other hand, if a track is newly generated, its trajectory from running the Kalman filter backwards in time is compared with all previously selected tracks. If a significant overlap is found between the new track and an existing track, both tracks will be assigned with the same ID, and the track with the higher score will be kept.

3.5 Multi-sensor Tracker Architecture

Our pipeline implements the aforementioned MDL tracking algorithm in the ROS environment. One advantage of the MDL tracking algorithm is its independence from a specific sensor type. Thus our MDL tracker is able to utilize detections from all available sensors on a robotic platform. Detections made from different types of sensors naturally contain different uncertainties. This difference in uncertainty is implicitly handled by the Kalman filter in the MDL tracker.

The front-end of our tracker is a detection preprocessing module, which performs spatial and temporal fusion of detections from different sensors. For the spatial fusion, this module uses the known sensor calibration to transform all detections into a common coordinate frame. Since the sensors may have overlapping fields of view, duplicated detections are checked and suppressed. For the temporal fusion, this preprocessing module adjusts the asynchronous detections (the frequencies of sensors are different) into a unified time. Usually there will be a difference between this unified time and the time at which the detection is made. This difference is reflected in terms of uncertainty, i.e., the greater this difference is, the more the uncertainty of the detection will be increased.

This preprocessing module can either be data-driven or time-driven (**Figure 7**). For the data-driven preprocessing, the fusion happens for every fixed number of detections. Thus, the time interval in-between multiple fusions is not uniform, and is handled by the motion model in the Kalman filter. The time-driven preprocessing, on the other hand, performs fusion at a fixed frequency. In practice, the optimal driven approach depends on the number of sensors and their frequencies. The fused detections from multiple sensors are then passed into the MDL tracker and are tracked in a common coordinate frame.

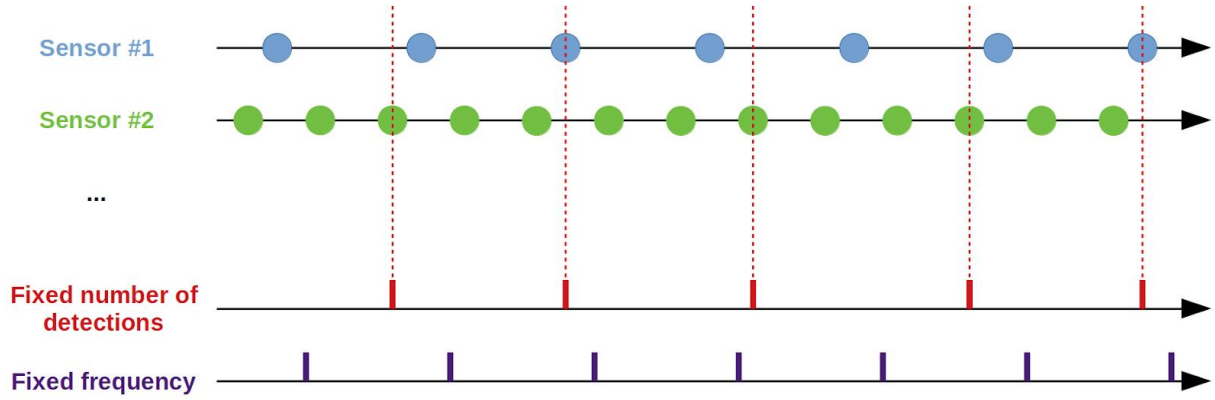


Figure 7. We perform spatial and temporal fusion of detections from multiple sensors. The fusion is carried out either 1) for a fixed number of detections, or 2) at a fixed temporal frequency. The fused detections are then passed into the MDL tracker.

4. Data Acquisition and Evaluation

Since the tracker was initially designed to deal with sparse to medium density crowds, a more realistic dataset is necessary to understand its capabilities and the potential challenges that a high-density crowd could pose for the tracking system. Moreover, having a dataset that is recorded within the CROWDBOT project setup is extremely important so as to simulate the robotic environment used there, and to evaluate the tracker quantitatively. This will also help us to have some baseline which can be used as a reference to incorporate any enhancements to the tracker pipeline.

4.1 UCL Recording

To leverage the benefits from having such an internal dataset, we performed a data capture session organised in a collaboration of RWTH and UCL at the UCL Stanmore campus in London with an intention to capture a test dataset, using the wheelchair robot, that covers some of the scenarios relevant to the CROWDBOT project.

Setup: This experiment was organised at the Aspire student centre in UCL's Stanmore campus. The data capture dealt with only indoor scenarios, where a corridor was simulated, and some voluntary participants were made to walk around. The wheelchair robot was operated by a person who followed these participants, getting as close to them as possible, and moving along with these participants simulating a wheelchair user navigating in such a closed environment. A detailed list of test scenarios covered in this data capture is shown in Table 1.

4.2 Annotation Process

As one of our intentions in recording an internal dataset was to evaluate the tracking pipeline, it became necessary to annotate the recorded dataset. These annotations will serve as ground-truth tracks against which the output of our tracking pipeline can be compared. Since we are interested in bounding box level detections and tracks, the objects in each of the frames are annotated with the corresponding bounding boxes.

4.3 Annotation Strategy

The focus of the tracking module within the CROWDBOT project is to detect and track people, which enables autonomous navigation through dense crowds. For this reason, we annotate every person in each of the video sequences ignoring other object classes. Thus, any object that is not a person is considered as background.

The annotation strategy is the same for every person in the video which represents a single group. A person who is in the field of view is annotated irrespective of whether he or she is moving, standing, bending over, or sitting. The bounding box contains all pixels that represent the object, and is bounded by the object's shape as accurately as possible. If a person is occluded by other objects, then the extent of the bounding box represents an estimate of the person's full extent including the part that is occluded. Occlusions are not marked and the occluded objects are fully annotated with a bounding box.

Scenario	Description	Camera Mount Height	Crowd Flow
1	Wheelchair is stationary, and people move in front of it.	ca. 152cm	1D Flow, 2D Flow
2	Crowd is initially stationary, and the wheelchair moves towards it. Once the wheelchair gets close to the crowd, the crowd starts to move in one direction and the wheelchair follows it.	ca. 152cm	1D Flow
3	The crowd moves both towards and against the wheelchair, while the wheelchair tries to navigate through the crowd. People often try to overtake the robot and switch direction abruptly.	Ca. 152cm	1D Flow, 2D Flow, Cross Flow
4	Same as scenario 3 but with a lower camera mount height and a different wheelchair user.	ca. 146cm	1D Flow, 2D Flow, Cross Flow
5	The wheelchair moves in a direction orthogonal to the crowd, cutting across the crowd flow.	ca. 146cm	Cross Flow

Table 1. Test Data Scenarios.

Each person is assigned an ID which represents the track ID, and remains the same until the end of the trajectory. A trajectory is defined by the time when a person first enters the field of view and the time when he/she leaves. If a person reappears soon enough for any tracking system to be able to learn the motion model, then the person gets the same ID. If he or she reappears at a later point of time, then a new track ID should be assigned. The decision on whether a person re-appears soon enough to continue a particular trajectory is left to the discretion of the annotator.

4.4 Sequences

Based on the above annotation rules, we have fully annotated the first four sequences that were recorded (represented by Scenarios 1-4 in Table 1). The annotation for the fifth scenario is an ongoing work and is expected to finish soon. Table 2 shows the statistics of the sequences that are annotated. Annotated objects represent the number of bounding boxes that are annotated, while annotated frames shows the total number frames annotated for a particular sequence.

A few annotated frames showing the detections along with the corresponding track IDs are shown in Table 3. It can be observed that every person is annotated with a bounding box that is tightly bound to the object, and every object gets a track ID which is denoted at the top left corner. It can also be observed that people who are partially visible are annotated with the full extent based on a shape prediction made by the annotator.

4.5 Data Format

We follow the MOT16 [Milan16] data format for both the annotations as well as the tracker output. In this format, the data is represented by a comma separated csv file with 9 values. The first value represents the frame number in which the corresponding object instance occurs; the second value denotes the track ID, which is a unique ID used to represent the object trajectory; and the next four numbers form the bounding box representation. A bounding box is represented by the spatial location of the top left corner and the corresponding width and height. The last two values are ignored and set to -1. Each entry in the detection file represents a single object instance. For more details on the MOT16 data format, please refer to the original paper [Milan16].

Scenario	Length (sec)	#Frames	Annotated Objects	Annotated Frames
1	243	2402	6999	2401
2	220	3648	4973	1632
3	283	3426	3099	1274
4	435	2793	3901	1986
5	169	1199	-	-

Table 2. Annotation statistics.

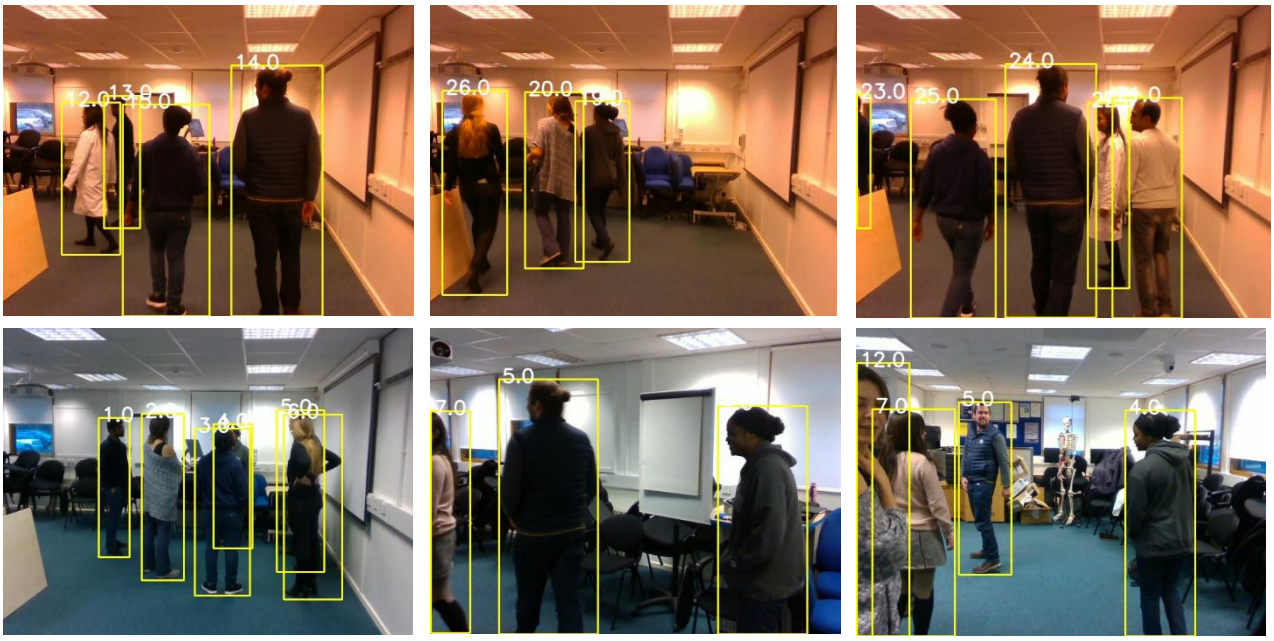


Figure 8. Annotation samples from Scenario 1 (top) and Scenario 2 (bottom).

4.6 Evaluation Criteria

To evaluate our detection and tracking pipeline, we first evaluate our detector, and then the fully integrated tracker. Following common conventions, we use average precision (AP) as the main measure for evaluating our detector (YOLO), and the metrics used in the MOT16 [Milan16] for evaluating the tracker. Specifically, we use multi object tracking accuracy (MOTA) as the main measure to compare the tracker's performance, and a set of other specific performance estimates such as false positives (FP), false negatives (FN), id switches (IDs), mostly tracked (MT), mostly lost (ML) etc. For a detailed description of these metrics, please refer to the MOT16 [Milan16] and CLEAR 2006 evaluation [Bernardin08] papers.

5. Results

5.1 Quantitative Results

All evaluations are performed on the UCL-CROWDBOT datasets. We use only the first four sequences since they are fully annotated, and leave the evaluation on the fifth sequence for future work.

Detector	Input Size	mAP	FLOPS	FPS	Model Size
YOLOv3	416x416	55.3	65.86 Bn	35	237 MB
YOLOv3-tiny	416x416	33.1	5.56 Bn	220	4 MB

Table 3. Accuracy and Run-time performance of different YOLO versions on the COCO test-dev set.

Scenario	Detector	AP	Recall	mIOU
1	YOLOv3	78.44	85.02	77.33
2	YOLOv3	71.35	83.14	79.87
3	YOLOv3	69.55	85.47	77.75
4	YOLOv3	70.45	87.96	75.58
1	YOLOv3-tiny	38.34	45.49	79.68
2	YOLOv3-tiny	43.33	49.74	83.40
3	YOLOv3-tiny	42.59	51.91	81.92
4	YOLOv3-tiny	41.63	51.72	79.48

Table 4. Detector evaluation on the UCL Wheelchair dataset.

5.2 Detection Rates

For evaluating the detector, we compare the average precision on the UCL-CROWDBOT dataset for YOLOv3 as well as the YOLOv3-tiny detectors. YOLOv3-tiny is a stripped down version of the full YOLOv3 detector, which is designed to run very fast with a small resource set. YOLOv3-tiny uses a small network and can run at a speed of 0.16 sec per image on a CPU. A comparison of performance and speed of YOLOv3 and YOLOv3-tiny is given in Table 3. Both variants that are used for experiments in this section take as input an image of size 416 x 416. It can be observed that YOLOv3-tiny runs at a very high speed with a small model that consists of 3x3 and 1x1 convolutions at a significant cost of detection accuracy.

Table 4 shows the results of running YOLOv3 and YOLOv3-tiny on the first four sequences of the UCL-CROWDBOT dataset. On top of average precision, we also report the average intersection over union (IOU) of the detected bounding boxes with the ground truth. It can be seen that the performance of YOLOv3 is significantly better in terms of average precision showing the superiority of a larger network. It is also interesting to see that the mean IOU of the detected bounding boxes with the ground truth is slightly better for YOLOv3-tiny. This could be due to the fact that the recall for YOLOv3-tiny is very low, and hence it produces detections only for those objects which are fully visible. It is worthwhile to note here that we do not penalize missed detections during the IOU calculation. Another observation from these results is that the YOLOv3 detector performs well on Scenario 1, where the camera is static, while performing relatively worse with larger camera movements, as can be seen in the other three scenarios. YOLOv3-tiny on the other hand performs better in scenarios where the object is closer to the camera as is the case in Scenario 2, 3 and 4.

5.3 Tracking Accuracy

In previous publications [Osep17], we have already demonstrated that our multi-object tracking approach achieves state-of-the-art performance on public benchmarks such as KITTI [Geiger12]. In this section, we therefore focus on measuring the tracking performance of the current tracker configuration and parameters used within CROWDBOT, such that we have a way of systematically optimizing the tracker parameters for the target scenarios.

Scenario	Detector	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDS	MOTA	MOTP
1	YOLOv3	99.0	77.1	111	18.6	89	3	1089	1734	508	35.4	80.9
2	YOLOv3	95.1	66.6	34	5	26	2.3	534	1031	265	38.9	80.5
3	YOLOv3	92.8	85.1	88	19	57	11	409	817	294	29.9	88.1
4	YOLOv3	88.6	82.3	146	43.5	81	21	678	768	252.5	33.9	85.0

Table 5. Tracker results on the UCL wheelchair dataset using the CLEAR MOT evaluation metrics from [Bernardin08]. (Rcll = bounding box recall; Prcn = bounding box precision; GT = #ground-truth tracks; MT = number of "mostly tracked" tracks (>80% of track length); PT = "partially tracked" (between 20% and 80% of track length); ML = "mostly lost" (<20% of track length); FP = #false positives; FN = #false negatives; IDS = #ID switches; MOTA = "Multi-Object Tracking Accuracy"; MOTP = "Multi-Object Tracking Precision").

	GeForce 1080Ti	Jetson Tx2	AGX Xavier
Power Requirements	250W	7.5W / 15W	10W / 15W / 30W
YOLOv3 Detector	20-30 fps	3 fps	7 fps
+TensorRT	-	-	20+ fps
Detection + Tracking	11 fps	3 fps	7 fps
+TensorRT	-	-	10+ fps

Table 6. Detector/tracker run-times and power consumption on different GPUs.

For evaluating the tracker performance, we use the full tracker pipeline with a YOLOv3 detector run on an Nvidia GeForce 1080 GPU. Table 5 shows the result of running the complete tracking pipeline on the first four sequences of the UCL CROWDBOT dataset. As can be seen, the tracker has a high track recall and gets a MOTA score between 30 and 40 on all the sequences. This is a reasonable baseline score considering the fact that the UCL CROWDBOT dataset is a set of challenging sequences with dense crowds, large camera motion, and heavy occlusions, and that neither odometry data nor a precise camera calibration was available for the recordings.

The current version of the tracking pipeline requires a GPU to run in real-time. Since a proper desktop GPU requires a lot of power, and a GPU based system occupies quite some space, it is too costly to use them in mobile robotics scenarios. Hence, we also carried out a performance test of our tracking pipeline on several NVidia Jetson platforms. The NVidia Jetson is an embedded GPU accelerated platform offered by NVidia, and is suited for mobile systems since it can work with low power. The run-time performance results of the tracking pipeline are given in Table 6, where we evaluate both the detector and the integrated tracker separately.

It can be seen that the detector runs at its peak performance with a GeForce 1080Ti GPU, which is a proper desktop GPU, and the tracker subsequently can run at around 11 fps. When we switch to the embedded platforms (Jetson TX2 and AGX Xavier), the computational resources limit the performance of YOLOv3, which in turn is reflected in the tracking speed. In addition to this, we also run a performance test on the tracker with TensorRT optimisation on the AGX Xavier board. TensorRT is NVidia's optimised inference engine, which is developed specifically to optimise deep learning inference. With this optimisation in place, the YOLOv3 detector runs at around 20 fps which is more than twice as fast as it can run normally on the AGX Xavier board. This optimization also results in the tracker running at a speed of 10 fps, which is close to its performance on an NVidia GeForce 1080 Ti GPU.

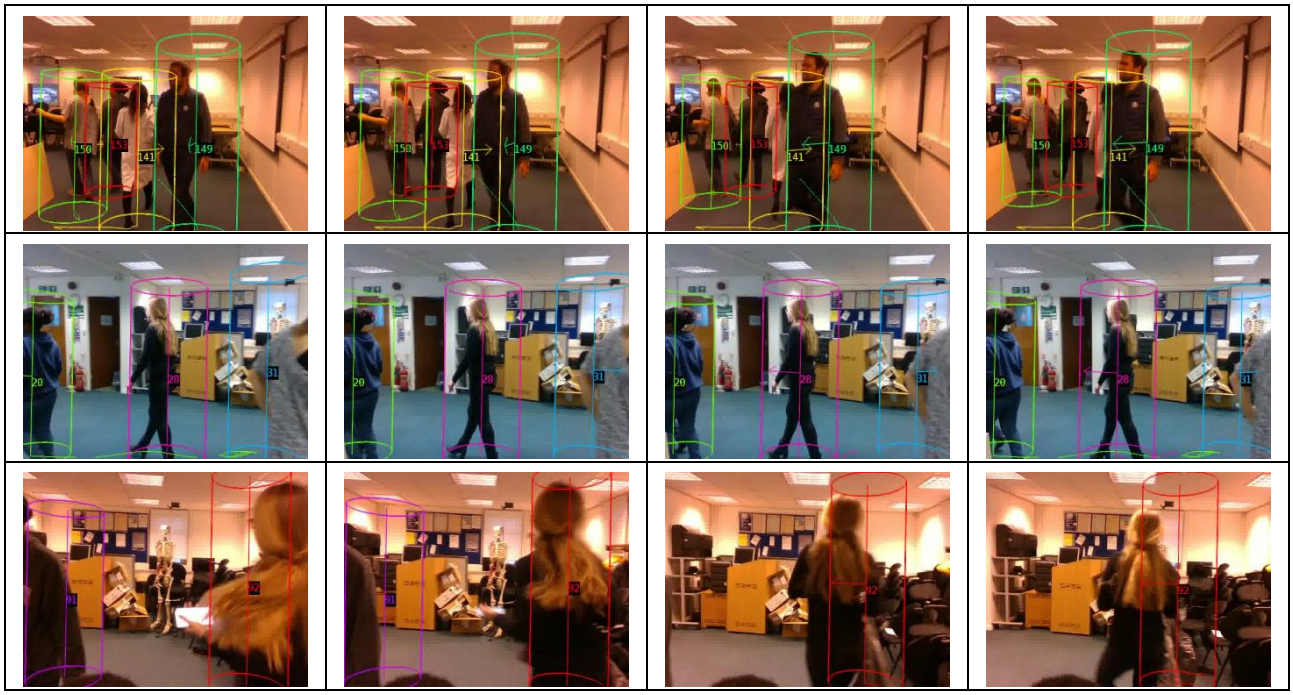


Figure 9. Tracker result on the UCL Wheelchair dataset. (top row) Medium density crowd in constrained space; (middle row) sequences with objects of partial visibility; (bottom row) camera mounted at a low height.

5.4 Qualitative Results

Some sample video frames, along with the output of the modified tracking system run on these frames are shown in Figure 9. As can be observed, the tracker already performs well on the crowd density captured in this experiment. The first row shows the tracker results on a moderately crowded scene for Scenario 1, where the tracker successfully tracks even occluded objects. The second row shows some frames from Scenario 3, where the detector successfully detects objects with partial visibility and the third row shows the tracker running on a low mounted camera setup with a part of the wheelchair user's head visible.

6. Extensions and Ongoing Work

The tracking pipeline was originally designed to work with low to medium density crowds. Hence, it would be further useful to augment the tracker to improve its performance so as to reduce the failure probabilities while dealing with large density crowd scenarios. In this section, we discuss some of the planned extensions to the perception module, as well as some related work.

6.1 Optical Flow Augmented Tracker

When the crowd density becomes very large, conventional tracking-by-detection systems based on RGB data might fail due to heavy occlusion. In such cases it would be helpful to fall back to low level vision techniques such as optical flow or scene flow. Optical flow provides motion information at the pixel level for a small neighborhood within an image. This information can be useful to make motion predictions in cases where the tracker fails due to occlusions, for example when people close to the camera obstruct the field of view.

For this purpose, we modify the tracking pipeline by incorporating optical flow. In this first prototype, we augment the tracker by adding two additional components and by further updating the tracker engine to use the new information. These components are described below.

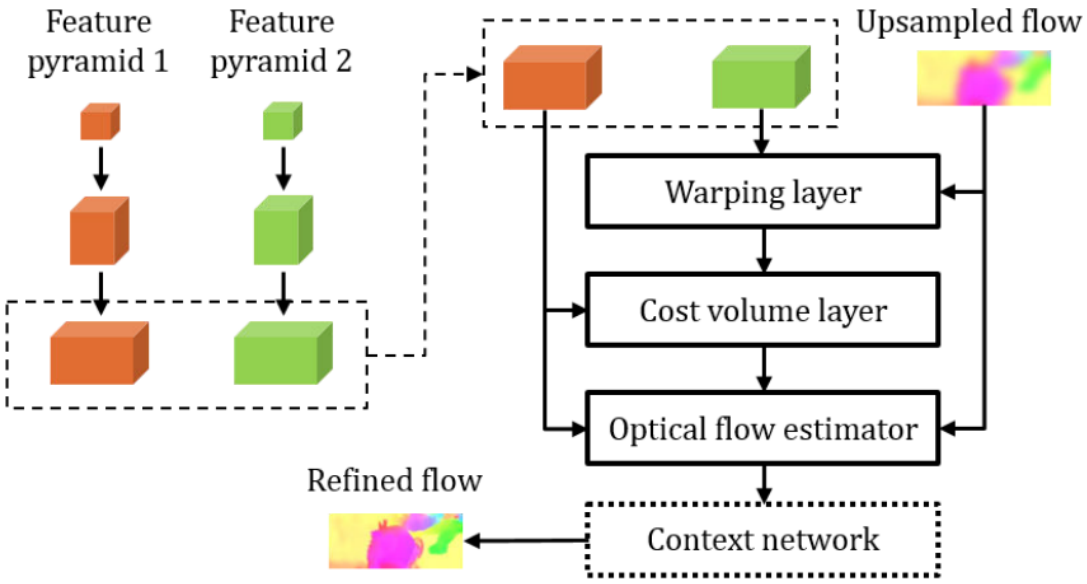


Figure 10. PWC-Net Architecture [Sun18].

6.1.1. PWC-Net

For computing optical flow between two images, we use the state-of-the-art PWC-Net [Sun18]. It is a deep learning based algorithm which takes two images (subsequent frames of a video), and generates the corresponding optical flow as output. PWC-Net is 17 times smaller in size and easier to train than the previous state-of-the-art FlowNet [Fischer15] model. Moreover, it outperforms all previously published optical flow methods on the MPI Sintel final pass and KITTI 2015 benchmarks, running at about 35 fps on Sintel resolution (1024x436) images. The speed and performance of PWC-Net makes it a very good choice for our tracking pipeline.

The architecture of the deep network that PWC-Net uses is shown in Figure 10. Main features of the network are pyramidal processing, warping, and the use of a cost-volume layer. It takes two frames as input, and computes feature pyramids for each of them. The features of the second frame are then warped onto the first frame using a low resolution flow prediction, which is then forwarded to the cost-volume layer that computes correlation between the feature maps. For further details, please refer to the PWC-Net paper [Sun18].

The pwc-net ROS node is a ROS wrapper for PWC-Net, which takes two subsequent images from the RealSense RGB-D camera and generates the corresponding optical flow vectors. We then warp the second image onto the first based on the estimated flow. Let us denote the input images as I1 and I2, and the warped image as I2w. We further use I2w to compute a confidence map by calculating the L1 distance between all pixels of I2 and I2w. This results in a confidence score that expresses how reliably the motion model can estimate the actual motion in certain image regions. A low confidence at certain pixels suggests that these regions could either be occluded or that the magnitude of the motion could be too large.

6.1.2. Addwarp Node

The second component in our optical flow augmentation is the addwarp ROS node. This node takes the confidence map from the pwc-net node and the bounding box information from the YOLO network and computes a score for each of the bounding boxes. This operation gives information about how confident the motion model can be, for the pixels within these detections. The lowest score would usually be for the most occluded regions. This is also the case in the border regions of the frame, because the network does not have prior information about regions outside the frame and it would be difficult to encode the motion in these regions. An updated architecture of the tracker after incorporating these new nodes is shown in Figure 11.

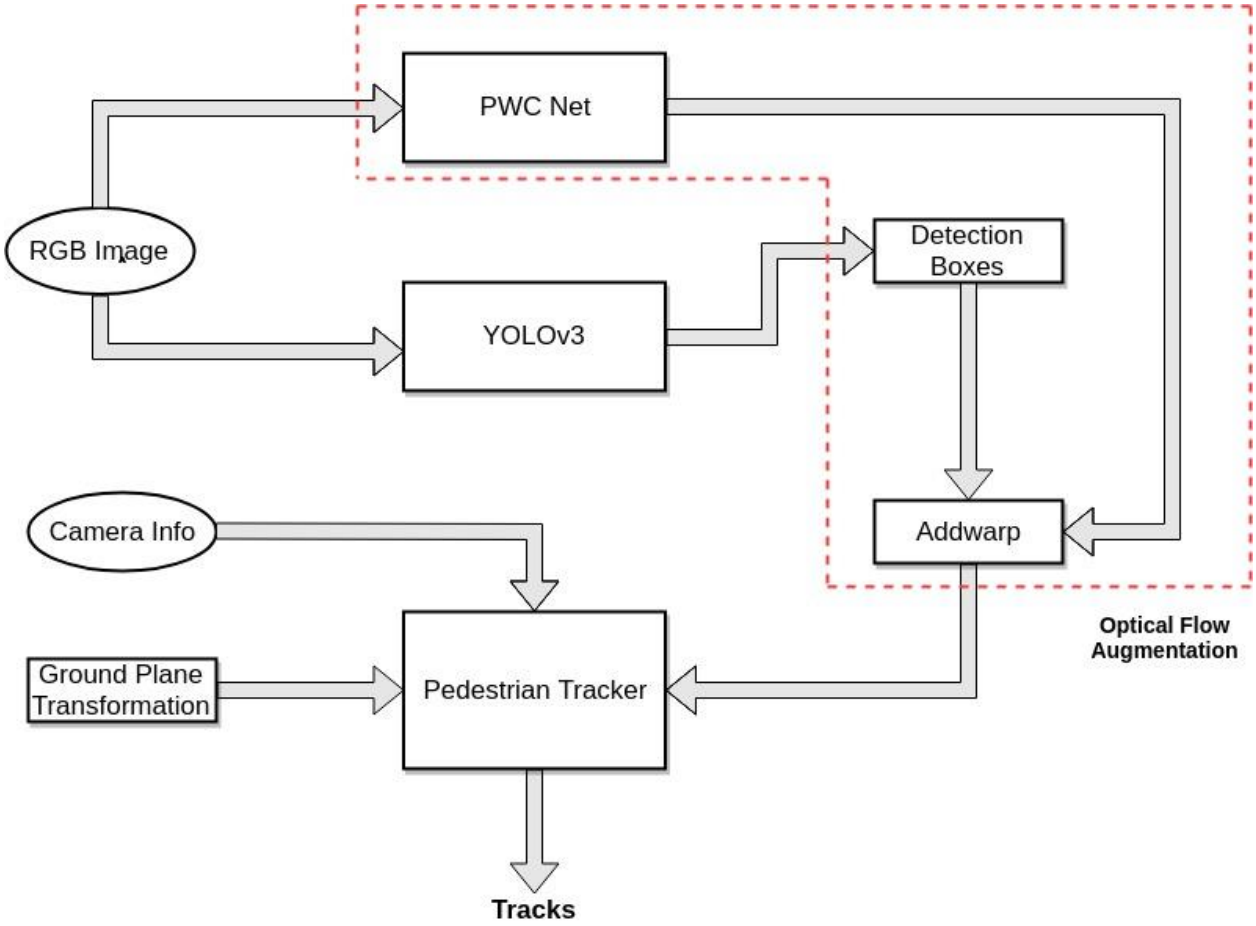


Figure 11. Optical Flow Augmented Tracker.

The confidence map is used in the tracker to decide whether to reject a bounding box or not. This would essentially happen if the confidence score is too low, e.g., in regions where the motion prediction is difficult. This should reduce the number of ID switches, and hence in better performance of the tracking algorithm.

6.1.3. Results

Table 7 shows the tracker performance on the UCL CROWDBOT dataset with optical flow augmentation. It can be seen that there is a slight advantage in terms of MOTA, except for Scenario 2, even if it is not very significant. In particular, optical flow seems to help in reducing the false positive and false negative scores in addition to providing a small gain in the "mostly tracked" (MT) trajectories. The effect of optical flow is not very significant due to the fact that it currently has an effect only on small subregions such as the border area, and other occluded areas. It is worthwhile to note that we do not do any odometry based corrections, and hence large camera motions might generate noisy optical flow values.

6.2 Interactive Annotation [BMVC'18]

In the last year, we have published a paper titled "Iteratively Trained Interactive Segmentation" at the British Machine Vision Conference (BMVC) 2018 [Mahadevan18]. The paper introduces a deep learning based interactive segmentation algorithm which can help reduce the effort needed to annotate data. Annotated data is essential for evaluating a tracking pipeline, which in turn helps in tuning and enhancing trackers as we have already seen in the previous section. Apart from this, today's state-of-the-art detection and classification methods are based on neural networks, which require a huge amount of annotated data to train.

Scenario	Detector	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDS	MOTA	MOTP
1	YOLOv3	98.8	**78.7**	111	**20**	89	**2.6**	**1053**	**1719**	**500**	**36.4**	80.8
2	YOLOv3	95.1	63.0	34	**6**	26	**2**	554	1055	265	38.0	80.4
3	YOLOv3	**92.9**	**85.3**	88	**22**	57	11	**392**	**799**	**292**	**31.0**	**88.2**
4	YOLOv3	**88.7**	**82.9**	146	**44**	81	**20**	**671**	**753**	263	**34.1**	**85.1**

Table 7. Tracker results with optical flow augmentation. Values in bold denote an improved result compared to the tracker without using optical flow (for an explanation of the different columns, see Table 5).

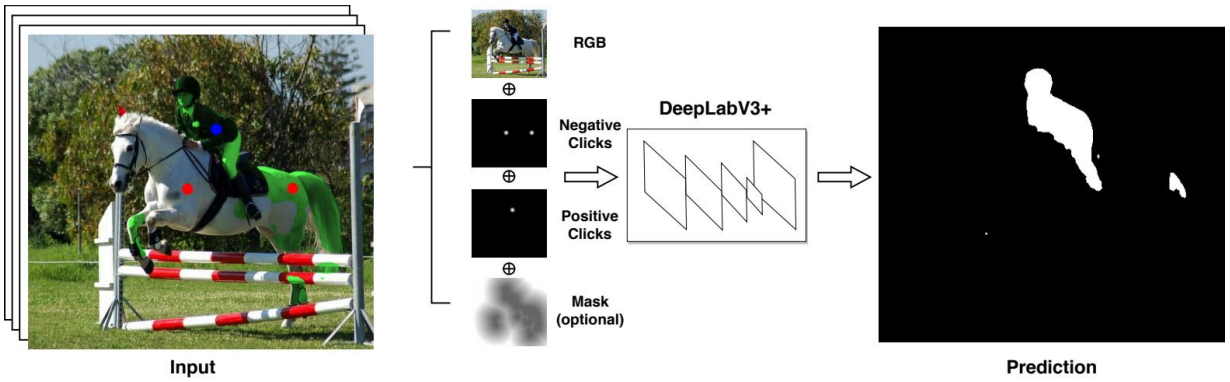


Figure 12. Overview of our interactive segmentation framework [Mahadevan18].

Our paper introduces a convolutional neural network based interactive segmentation algorithm, which accepts user inputs in the form of clicks, and generates the corresponding pixel level segmentation label. An overview of the method is shown in Figure 12. Our network, which is based on DeepLabV3+ [Chen18] takes an RGB image, along with two channels that represent the negative and positive user clicks, as input, and generates a pixel mask as the output. The clicks are encoded as Gaussians with a fixed scale, centered on the clicked location. In addition, the network also takes an optional mask channel as input in case the user has to refine an existing segmentation mask. This mask channel, if used, is encoded in the form of a Euclidean distance transform.

Our BMVC'18 paper also introduces a novel iterative training strategy, where synthetically created user clicks are added iteratively in the training stage, thereby simulating the usage of such an interactive system at test time. As part of this training procedure, a new click sampling strategy, that simulates a user's click refinement during annotation, was used to sample clicks for each iteration. We refer to our paper [Mahadevan18] for more details.

6.3 Detailed Person Analysis [GCPR'19]

Perception of people and of the robot's environment can further benefit from detailed person analysis tasks such as person re-identification, body part segmentation, or articulated body pose estimation. In the past, each of these problems was solved individually by learning different models for them. However, such an approach is too inefficient for mobile robotic applications, where GPU access time for deep learning modules is a major bottleneck. In addition, the separate task-specific networks could not benefit from each other, thereby making holistic scene understanding difficult. Multi-task learning tries to solve these issues by approaching the correlated estimation problems jointly. Specifically, a joint learning strategy is used to produce a single model that can perform multiple tasks simultaneously. Such a joint learning approach is extremely useful for robotic applications where the available resources are limited and hence computation costs have to be kept in check.

However, the devil is in the details. When training a network on multiple tasks simultaneously, it is not guaranteed that the results will actually improve – on the contrary, the forced feature sharing may lead to performance degradation. Further complications arise when no training data is available that is simultaneously annotated for all target tasks.

We have therefore conducted a thorough study of this problem for the use case of holistic person analysis by jointly learning multiple person related tasks, such as person re-identification, body pose estimation, attribute classification, and body part segmentation. This work led to a publication at the German Pattern Recognition Conference (GCPR) 2019 [Pfeiffer19] and is highly relevant to the CROWDBOT project. Our proposed approach uses a common CNN backbone network and multiple prediction heads for the different detailed person analysis tasks. Figure 13 shows several single- and multi-head backbones that we explored in our study to learn different task combinations. In addition to a joint learning model, our approach also effectively combines multiple datasets to learn the individual tasks, since a single annotated dataset does not exist for all of them. Example network predictions are shown in Figure 14. As can be seen, given the input frame and the corresponding person detections, the CNN generates pose estimation, body part segmentation and attribute classification. Person re-identification is not visualized here. Please refer to the paper for details ([Pfeiffer19]).

Since it is a relatively recent development, the detailed person analysis is still in the development stage and has not been integrated into the tracker yet. In particular, further research is needed in order to explore the best way to incorporate the detailed person analysis heads with the YOLO network used for object detection.

7. Conclusion

In this report we have presented the first prototype of the perception pipeline designed for the CROWDBOT project, which currently focuses on detection and tracking of pedestrians in low to medium density crowds. The pipeline is flexible with respect to the sensor configuration and can be easily adapted to all robotic platforms within the CROWDBOT project. Evaluations on multiple components along the perception pipeline were performed using data collected in an RWTH-UCL collaboration, with very promising results. Multiple extensions, including detailed person analysis and crowd flow estimation, are in progress to further extend the capabilities of the perception pipeline.

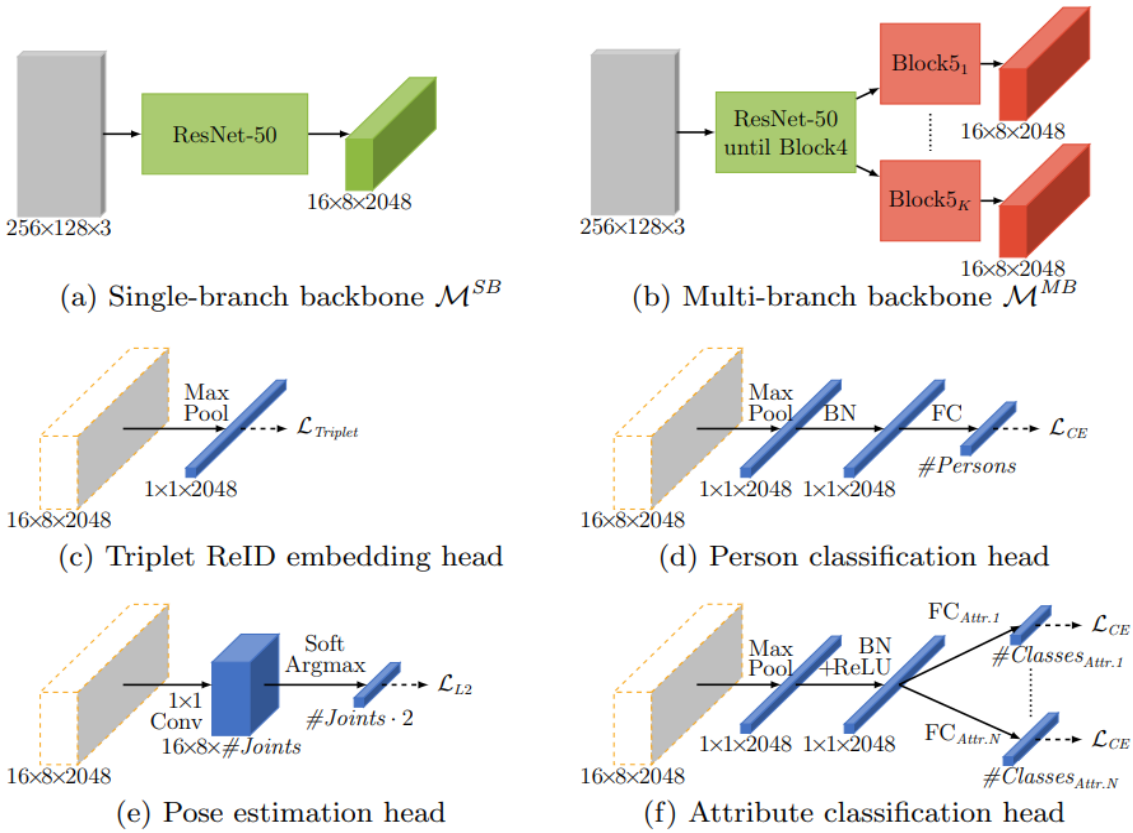


Figure 13. Backbone networks with single and multiple heads [Pfeiffer19].

Figure 14. Sample outputs for pose estimation, body part segmentation and attribute classification [Pfeiffer19].

References

(entries in bold denote publications made within CROWDBOT)

[Bernardin08] K. Bernardin, R. Stiefelhagen," Evaluating multiple object tracking performance: The CLEAR MOT metrics", EURASIP Journal on Image and Video Processing, 2008.

[Beyer17] L. Beyer, A. Hermans, B. Leibe, "DROW: Real-Time Deep Learning-Based Wheelchair Detection in 2-D Range Data", IEEE Robotics and Automation Letters, Vol. 2(2): 585-592, 2017.

[Beyer18] L. Beyer, A. Hermans, T. Linder, K.O. Arras, B. Leibe, "Deep Person Detection in Two-Dimensional Range Data", IEEE Robotics and Automation Letters, Vol. 3(3): 2726-2733, 2018.

[Chen18] L-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation", arXiv 1802.02611v3, 2018.

[Dalal05] N. Dalal, B. Triggs, "Histograms of Oriented Gradients for Human Detection", In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

[Fischer15] P. Fischer, A. Dosovitskiy, E.Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. Smagt, D. Cremers, T. Brox, "FlowNet: Learning Optical Flow with Convolutional Networks", IEEE International Conference on Computer Vision (ICCV), 2015.

[Geiger12] A. Geiger, P. Lenz, R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[Jafari14] O.H. Jafari, D. Mitzel, B. Leibe, "Real-Time RGB-D based People Detection and Tracking for Mobile Robots and Head-Worn Cameras", In IEEE International Conference on Robotics and Automation (ICRA), 2014.

[Leibe08] B. Leibe, K. Schindler, N. Cornelis, L. J. V. Gool, "Coupled object detection and tracking from static cameras and moving vehicles", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 30(10), 2008.

[Mahadevan18] S. Mahadevan, P. Voigtlaender, B. Leibe, "Iteratively Trained Interactive Segmentation", British Machine Vision Conference (BMVC), 2018.

[Milan16] A. Milan, L. Taixe, I. Reid, S. Roth, K. Schindler, "MOT16: A Benchmark for Multi-Object Tracking", arXiv:1603.00831, 2016.

[Osep17] A. Osep, W. Mehner, M. Mathias, B. Leibe, "Combined Image- and World-Space Tracking in Traffic Scenes", IEEE International Conference on Robotics and Automation (ICRA), 2017.

[Pfeiffer19] K. Pfeiffer, A. Hermans, I. Sarándi, M. Weber, B. Leibe, "Visual Person Understanding through Multi-Task and Multi-Dataset Learning", German Conference on Pattern Recognition (GCPR), 2019.

[Redmon16] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[Redmon18] J. Redmon, A. Farhadi, "YOLOv3: An Incremental Improvement", arXiv:1804.02767, 2018.

[Reid79] D.B. Reid, "An Algorithm for Tracking Multiple Targets", IEEE Transactions on Automatic Control, Vol. 24, pp. 843-854, 1979.

[Schindler06] K. Schindler, U. James, H. Wang, "Perspective n-view Multibody Structure-and-motion through Model Selection", In European Conference on Computer Vision (ECCV), 2006.

[Sudowe11] P. Sudowe, B. Leibe, "Efficient Use of Geometric Constraints for Sliding-Window Object Detection in Video", International Conference on Computer Vision Systems (ICVS), 2011.

[Sun18] D. Sun, X. Yang, M. Liu, J Kautz, "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume", In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.