6-18-2024

Corridor Counting

Anthony Bryson

Vincent Zhou

Amy Ha

SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 18, 2024

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Anthony Bryson
Vincent Zhou
Amy Ha

ENTITLED

Corridor Counting

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

David C. Anastasiu

Digitally signed by David C. Anastasiu
Date: 2024.06.18 09:51:18 -04'00'

Thesis Advisor

Department Chair

Corridor Counting

by

Anthony Bryson
Vincent Zhou
Amy Ha

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 18, 2024

Corridor Counting

Anthony Bryson
Vincent Zhou
Amy Ha

Department of Computer Science and Engineering
Santa Clara University
June 18, 2024

ABSTRACT

Information on traffic patterns is essential for identifying and addressing sources of traffic congestion and informing future road layouts to create safer and more efficient roads. To this end, we develop a Corridor Counting, or Multi-Camera Vehicle Counting, algorithm that quantifies the number of vehicles traveling along a specific stretch of road. Our work builds upon the related problem of Multi-Camera Vehicle Tracking and draws inspiration from methods used for Single-Camera Counting. We propose a six-step solution comprising Vehicle Detection, Feature Extraction, Single-Camera Vehicle Tracking, Re-Identification, Movement Matching, and Multi-Camera Vehicle Counting. Finally, we adapt an evaluation metric from Single-Camera Vehicle Counting to assess the effectiveness of our solution.

ACKNOWLEDGMENTS

We would like to thank our advisor, Dr. Anastasiu, for being an essential source of motivation and assistance throughout our senior project.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Traffic congestion is a significant issue for drivers in busy cities around the world; not only does congestion waste valuable time, but it also results in more accidents. However, urban planners need to understand traffic congestion before attempting to address it. Current solutions for identifying and tracking vehicles using multiple video surveillance cameras at intersections in a road network are insufficient. To this end, we seek to identify and utilize state-of-the-art computer vision techniques to label vehicles in a road network accurately.

Our problem, "Corridor Counting," shares similarities with two existing AI and computer vision focuses: Multi-Target Multi-Camera Tracking (MTMCT), or more specifically Multi-Camera Vehicle Tracking (MCVT), and Vehicle Counting. MCVT involves tracking the movement of and re-identifying a target, in our case vehicles, across several cameras and perspectives. It has received much attention due to its applications in transportation, self-driving vehicles, and traffic analysis. Vehicle Counting, or Counting for short, is the tracking of vehicle movements as they pass through an intersection, specifically identifying the entrance and exit the vehicles make from the intersection. Our Corridor Counting system aims to build upon solutions to these two problems to track and count the number of vehicles that travel along specific stretches of road.

Combining these two technologies is critical for scaling traffic management systems that mitigate traffic congestion. The results obtained from this project will provide insight into traffic demand and vehicle ratio on individual roads, which can be used to improve the timing of signals in intersections and mitigation strategies for traffic congestion. In addition to enhancing existing infrastructure, Vehicle Tracking is crucial in intelligent city planning and crowd analysis. It helps engineers design better roads that optimize the flow of traffic in cities and create solutions that improve the safety of both drivers and pedestrians.

1.2 Project Proposal

The goal of our Corridor Counting project is to design an algorithm that can count the number of vehicles that travel along a pre-specified set of corridors. Our solution will involve tracking vehicles across multiple cameras positioned at intersections to determine an accurate sequence of turns performed. Using these sequences, we can deduce the path taken by each vehicle through the city and determine if it followed one of the corridors of interest.

As mentioned earlier, the algorithm itself builds upon existing fields in AI and computer vision to perform some of the tasks involved in this problem. So our work focuses on building upon an existing MCVT algorithm, taking inspiration from Counting algorithms to determine which movements each vehicle makes, and then eventually match those to a path which can be compared with the pre-defined corridors.

Chapter 2

Project Requirements

In this chapter we describe the functionality requirements for our project as well as requirements for what systems it needs to be compatible with.

2.1 Functionality

Our algorithm takes the videos from the CityFlowV2 [1] dataset as input, as well as definitions for each corridor of interest and camera specific annotations, then outputs the number of vehicles that completed each corridor during the span of the videos.

Before starting we had to decide if we would make an online or offline algorithm. Online algorithms process the frames of the videos as they come in. This means they only have access to information up to the current time. Meanwhile an offline algorithm has the full duration of the video available to it at once, so it can jump around as needed. Our decision would mainly affect the Re-Identification portion, which is discussed in Section 3.4. Ultimately we decided upon using an offline algorithm, as this was the more common approach and would allow us to focus on the Corridor Counting aspect of the project over adapting components to satisfy the online approach.

2.2 Compatibility

Since we were utilizing the WAVE HPC [2] as our running environment, our design had to be compatible with and optimized to run on it. Specifically, the WAVE HPC nodes are limited to 2 GPUs per node with 384 GB RAM on each node. Additionally, the system had a number of built in technologies and libraries, and any additions to this would require administrative privileges. Docker [3] was among the technologies not included, meaning that any baseline using Docker was not available for consideration. However, for many dependencies that were available through the workload manager SLURM [4], a fairly recent version was usually available. For example, Python was regularly updated to its most recent version, which ensured that we were hardly limited in more critical areas such as Python major version.

Chapter 3

Literature Survey

This chapter will provide an overview of existing literature on AI models and strategies relating to our problem. Our solution to the Corridor Counting problem utilizes these models as a basis and cleverly connects them to track and count vehicles across multiple cameras.

3.1 Vehicle Detection

The first step in the Corridor Counting process, Vehicle Detection, is frequently handled by a variety of deep learning based models. The most commonly used model was the YOLO family (namely v3, v4, v5, and v5n) [5, 6, 7, 8, 9, 10], with a few other models being used, including Faster R-CNN [11, 12] and NAS-FPN [13].

These deep learning models fall into one of two possible categories, one-stage detectors and two-stage detectors [10]. One-stage detectors, including YOLO and SSD, treat object detection as a regression problem. Because of this, they are able to perform detection on an image with one pass through the network. The result is fast inference times, citing 45 frames per second on the base network on a Titan X GPU [10, 14]. With two-stage detectors such as the R-CNN family and NAS-FPN, they first have a region proposal network to identify candidate detections, which are then fed into a classification and regression network [10, 15, 16].

One notable approach from 2007 presented in Chen et al. (2007) [17] used background based object detection, called background extraction, instead of deep learning. Background extraction and deep learning models are the most popular ideas presented for object detection, with the later being much more common in present day algorithms [12]. This is not to say contemporary implementations of Vehicle Detection do not use background extraction, as it is used in Chen and Lu [18] and Lance et al. [19]. However, regardless of the approach, the result of these algorithms is a list of bounding boxes around each vehicle in a frame.

In Bui et al. (2020a) [5], Bui et al. (2020b) [7], and Bose et al. [6], they use YOLOv3 to do the object detection. This model also performs classification of the objects it detects, allowing them to ignore any bounding boxes that do not fall into the categories of cars, trucks, or bikes. However, in Bui et al. (2020a) [5], they eliminate motorcycles

and other two wheeled vehicles. In Bui et al. (2020b) [7] and Bose et al. [6], they further optimize the accuracy of the detection algorithm by training it on the COCO and MS-COCO data sets, respectively. These are labeled data sets containing common objects, including cars, buses, trucks, motorcycles, and bikes.

In Nguyen et al. [8] they elected to use YOLOv4 with CSP (Cross Stage Partial Network) for the object detection phase. They selected it over other detectors, such as YOLOv3 with SPP and Faster R-CNN with ResNet-101, due to its higher performance on occluded vehicles, ones that are far away from the camera, and large scale differences.

In Kolleck et al. [9] they chose the YOLOv5n model to do the object detection. In order to improve the classification accuracy of this model, they trained it on the COCO, MIO-TCD, and CGMU data sets. Their goal was to make a Counting algorithm that could be performed in real time, on an embedded system with limited computational resources. The v5n detection model sacrifices some accuracy for a 59.1% improved inference time. This is further improved in their model by 35.2% through a conversion to half-precision floating point values.

In Liu et al. [11] they use Faster R-CNN as their object detector. Similarly to the previous models, they had it pre-trained on the COCO data set. However, since their model was created to compete in the 2020 AI City Challenge [20], they further tuned it with the AI City 2020 Track-1 data set.

Chen et al. [17] and Lance et al. [19] use a pure background extraction approach to object detection. This relies on the fact that the background of the videos, namely the road and surrounding environments, remain relatively constant throughout. Their method develops a background image by averaging out many frames from a video. The result is an image containing an empty road. From this, detection can be done by comparing future frames with the background image. Further more, their model performs classification of the objects by analyzing characteristics such as perimeter, area, and bounding box.

In Wang et al. (2020) [13] they implement a two fold approach to object detection. The first part is using the NAS-FPN deep learning model, which they found outperformed other algorithms, including YOLOv3, in a combination of effectiveness and efficiency. However, they note that deep learning based approaches to object detection face difficulties in extreme situations, such as during rain or when there are significant lighting changes. To address this, they integrated the second part, background extraction, to perform additional detection. These two parts work together to produce a more accurate detection than existing methods.

3.2 Feature Extraction

Information about the physical features of the vehicles identified by a Vehicle Detection algorithm can be very useful. Specifically, there are Single-Camera Vehicle Tracking and Vehicle Re-Identification (ReID) algorithms that make use of these features. Single-Camera Vehicle Tracking involves identifying the frame by frame connections between bounding boxes to create a full trajectory of a vehicle's movement across a camera perspective. Vehicle ReID is the

task of distinguishing between a given vehicle and other vehicles across multiple views from a set of cameras.

Vehicles of the same make and model have relatively few features that uniquely identify them. In some views, such as a head-on shot, a feature like a rear license plate may not be visible. To address this peculiarity with vehicles, many approaches divide the features of a vehicle into multiple channels. For example, in Zhang et al. (2022) [21], researchers used a dual attention granularity network consisting of a branch for coarse-grained similarity (e.g., vehicle model and color) and a branch for fine-grained similarity (e.g., windshield stickers and decorations). Each branch utilized self-attention to detect regions of interest and extract regional features, allowing them to have more feature spaces separately. This optimization reduced conflict and made the triplet loss more effective, significantly improving performance.

Similarly, Guo et al. [22] chose two branches for channel-pooling and spatial-pooling attention to extract and enhance locally important information about vehicles from unmanned aerial vehicle photography scenes, which often have far fewer local features.

In Jiang et al. [23], a global reference node is built and compared to all the nodes in an image. Their network is built using the global reference attention modules created by the global reference node, enabling many discriminative features to be mined for future ReID.

Many other approaches also use multiple self-attention mechanics learned from advancements in Natural Language Processing. In Zhu et al. [24], a similar dual self-attention mechanism to Zhang et al. (2022) [21] is used. The proposed module used a static self-attention branch to selectively refine semantic features and a dynamic self-attention branch to enhance local features' spatial awareness. This network allows the model to capture long-range dependencies and relative position information, leading to an effective vehicle feature embedding.

3.3 Single-Camera Vehicle Tracking

Single-Camera Vehicle Tracking, which creates trajectories for each vehicle on a camera, has two approaches, offline and online tracking. With offline tracking, the algorithm has access to all the frames and their bounding boxes at once and utilizes graphs to construct the trajectories. In online tracking, the algorithm only has access to the current and past frames. DeepSORT is far an away the most popular online tracking algorithm, being used in most papers on Counting [11, 5, 9, 13, 7, 8]. Some other noteworthy approaches to online tracking specifically for detection based Vehicle Counting include StrongSORT, an improvement of DeepSORT, being used in Ahmad et al. [25], intersection over union (IOU) tracking in Lance et al. [19], and a corner detection based approach in Al-Ariny et al. [26].

Nguyen et al. [8] utilizes DeepSORT as the main tracking module for their Counting solution. Specifically they used DeepSORT pre-trained for pedestrian tracking. With this model, they found it had trouble with ID switches, especially in scenes with high vehicle density and occlusion as well as when small vehicles, such as motorcycles,

were present. To address these issues they introduced a trajectory association module to do post-processing on the DeepSORT tracklets. This module attempts to link two tracklets together if their trajectories and positions are similar and they either started or ended before reaching the end of the ROI.

Ahmad et al. [25] details why they used StrongSORT over the more popular DeepSORT. DeepSORT performs tracking based on the motion and appearance of the vehicles. StrongSORT uses a stronger extractor for the appearance features to improve tracking accuracy. Additionally, it utilizes a ResNeSt50 backbone to extract more discriminative features. These improvements allow for better tracking, and as a result, better counting in the future steps.

In Lance et al. [19] they perform tracking through intersection over union (IOU). This is a metric that measures bounding box overlap of detected vehicles in consecutive frames. For a new frame, a candidate bounding box is compared with all the unmatched bounding boxes of the previous frame. If the highest IOU bounding box surpasses a certain threshold, they are matched together, otherwise it is considered a new vehicle. This approach worked well in this case due to the lack of significant occlusion of vehicles in the camera angles used in the videos.

Similarly, Ospina and Torres [12] took inspiration from the Tractor object tracking algorithm. The Tracktor algorithm, presented in Bergmann et al. [27], performs object tracking based on the bounding boxes produced by Faster R-CNN. Ospina and Torres found that Bergmann et al. had a faster inference time in their tracking than other algorithms, so they implemented the same detection-tracking pipeline for their Counting algorithm.

In Al-Ariny et al. [26] they implemented a corner based tracking algorithm based on KLT tracker. This works by using the corners of the bounding boxes of detected vehicles and measuring the change in coordinates of these corners between frames. Any corners that move out of the ROI are deleted to reduce the complexity of the tracking.

Zhao [28] describes a tracking algorithm using a CNN with the ImageNet-vgg-2048 network. The tracking is based on particle filtering since it is more robust than some other tracking algorithms. The output of a Feature Extraction network is used as input to a logistic classifier, which the particle filter tracking algorithm uses to track the vehicles.

Yu et al. [29] uses a similar approach to tracking, relying on Mask R-CNN to provide feature representation vectors for the detected vehicles. From this, all the new detections in a frame are assigned to existing tracklets based on their feature similarities and some spatial constraints.

3.4 Vehicle Re-Identification

Vehicle ReID is a technique that finds a given vehicle across multiple views from a set of cameras. In an intelligent transportation system, vehicle ReID is often used to identify traffic violations, making it especially useful in metropolitan areas. Because vehicle ReID and person ReID share many similarities, such as worries about lighting conditions or occlusions, optimizations that boost the performance of one often work for the other. However, vehicles can differ far more significantly than people in alternative views from the same or different camera.

In Xu et al. [30] they describe "Rev", a video engine that focuses on solving identification efficiently. It does this by retrieving locations and times a vehicle appears rather than all the images it was sighted in and making optimizations to how typically unreliable features are selected and sampled. They mention the common solution compares vehicle features to a given query by performing clustering on the vehicle features. This process can be expensive, and the insights the authors make are in removing the need to label all bounding boxes correctly and the idea that better results can be obtained from focusing on gathering data from a wide variety of cameras.

Other optimizations are focused on improving the training data available for vehicle ReID. Like many fields in computer vision, labeling annotation for vehicle ReID training datasets is time-consuming and does not scale well. Unsupervised domain adaptation (UDA) can transfer learned representation information from a labeled source domain to an unlabeled source domain; however, because the performance still depends on the representation learning of the source domain, the expensive annotation is postponed rather than solved. In Wang et al. (2023) [31], a few-shot, dual-branch UDA framework is designed, which learns few-shot source samples to improve performance in the target domain. Then, further optimizations in the source branch, including a DCLSR loss, and optimizations in the target branch, including pseudo-label refinement with DDPT, were used to improve results further.

Yao et al. [32] introduces two new techniques for improving the common solution for ReID. They describe a zone gate mechanism which classifies vehicles to a number of zones based on their bounding box location. They then use each vehicle's zones to get an idea of which direction the vehicle was traveling and deduce which intersections the vehicle can and can't appear on. This allows them to drastically reduce the matching space for each vehicle and improve accuracy and speed. They also introduce a time-decay strategy which incorporates elapsed time into the ReID. By incorporating information about physical distances between intersections they can adjust probabilities of two detected vehicles matching based on when they appear at each intersection.

Holla et al. [33] describes a different zone based approach to Yao et al. [32]. They divide cameras into zones based on their physical locations. These zones then determine which gallery set a vehicle's images are assigned to. From there any query vehicle is compared against all the gallery images in its corresponding zone to find a ReID match.

3.5 Movement Matching and Counting

Vehicle Counting is the process of counting the number of vehicles traveling in a specific direction along a section of road. When this takes place at an intersection, the step of identifying which direction the vehicle is traveling in, called Movement Matching, becomes non-trivial. Detection and density based Counting are the two main strategies for Vehicle Counting [34], with most papers describing a detection based method as their approach of choice [11, 5, 9, 6, 17, 12, 13, 7, 8]. A density based approach performs worse when compared to the detection based approach when large vehicles or perspectives are involved [35] and isn't capable of performing Movement Matching. Density Counting

algorithms have better relative performance in heavy crowds. Both strategies suffer from occlusion decreasing the accuracy of results [11]. This section will look at the existing research that uses these methods.

3.5.1 Detection Based Counting

The common approach for detection based Counting divides the process into three distinct steps. The steps are Vehicle Detection, Single-Camera Vehicle Tracking, and then Movement Matching and Counting, which is based on the trajectories from the previous steps [11, 5, 9, 17, 13, 7, 36].

In Liu et al. [11] they perform the Counting phase using a semi-automatic method. Given videos from a specific camera angle, they manually label the entrance and exits associated with each movement with a line. From there, their algorithm groups the tracked trajectories based on which of the entrance and exit lines they cross. Averaging these out for each movement of interest gives a single path for each movement. Now given a new trajectory, their algorithm matches it to one of the average trajectories based on its Euclidean distance.

In Yu et al. [29] they use a Counting method that is a little more manual than in Liu et al. Their algorithm requires a labeled region of interest (ROI) and pre-defined routes, each in the form of a series of connected points. From this, they perform Counting by comparing the trajectory a vehicle took while in the ROI with all the pre-defined routes. In their comparison approach, they weighed the average distance from the pre-defined routes, how much progress the vehicle makes along the route with each step, and how much the distance from a route changes throughout the trajectory. They found that combining these three metrics resulted in the best accuracy in their counts.

In Ospina and Torres [12] they use a simpler but similar approach to that of Yu et al. Instead of using and performing computations with the entire path taken by the vehicles, Ospina and Torres create vectors from the vehicles first and last bounding boxes from when they were in the region of interest. The regions of interest were manually labeled at the edges of the video near the entrances and exits to the intersection. By using vectors instead of full trajectories, it becomes quicker and simpler to assign a vehicle to a movement, as it can be done almost exclusively based on the direction of the vectors.

Wang et al. (2020) [13] describes a Counting algorithm that can distinguish between the lanes the cars are driving in. It takes the trajectories of all the cars in the video and performs K-means clustering on them. From this, any future trajectories can matched to a lane-specific movement.

Bui et al. [7] points out a problem with tracking over the entire trajectory of the vehicle, being an ID switch. To address this, they used a distinguished regions approach to Counting which reduces the required tracking range and avoids occlusions from overlapping vehicles. In this, they define a number of distinguished regions over the entrances and exits in which they will perform tracking. This way, they can determine where the vehicles enter and exit from, and perform the counting once the vehicle has reached an exit region .

A number of papers presented a simplified version of the Counting problem in which all the videos analyzed were

absent of intersections. This simplified the problem because all vehicles were traveling in one or two directions, never crossing paths. In effect, it removed the need to perform Movement Matching to determine which direction vehicles were traveling. This allowed for a few different approaches, such as in Al-Ariny et al. [26] which performed counting whenever a new, untracked vehicle was detected. Obviously this would not be usable for Corridor Counting because information on which direction the vehicle travels after first being detected is needed. However, some other approaches more similar to those used when intersections are present were applied to this simplified version [18, 25, 37, 38, 39, 40]. Namely a Counting algorithm in which a number of arbitrary lines, one per direction being travelled in the video, were used to determine when the vehicles were counted. If the center of the vehicle's bounding box passed the line, it was counted.

3.5.2 Density Based Counting

Zhang et al. (2017) [35] proposes an alternative to the detection based method, instead relying on a density based algorithm to avoid individual detection all together. They look to do Vehicle Counting with videos that are low frame rate, low resolution, high occlusion, and large perspective. As discussed in their paper, the density approach does not perform as well as the detection based approach on large perspective videos. To make up for this, their method combines a fully convolutional network (FCN) that transforms pixels into vehicle density maps and a long short-term memory network (LSTM) to improve accuracy by learning complex temporal dynamics. This method removes the need for individual Vehicle Detection thus reducing the error that results from the low frame rate and low resolution video.

Zhang et al. (2021) [34] further expands on this density based Counting approach, citing dense traffic as an obstacle for training detection based algorithms. This is due to the detection algorithms requiring bounding box labels on the training data, which is difficult to produce for the heavy traffic scenes. The approach presented in this paper contains two parts, a density map estimation and the spatial-awareness block loss. The density map estimation is aware of the varying scales of vehicles. The spatial-awareness block loss, which divides the frame into smaller blocks to decrease the loss, helps balance out the high loss in regions with high density and occlusion with the low loss of the sparse regions. This allows the algorithm to get more accurate information in occlusions. In their testing they found their algorithm, VCNet, performed better than state of the art detection based Counting algorithms, especially with scenes containing congested traffic and occlusions.

As this approach to Counting does not perform individual detections, it is not too useful for our Corridor Counting pipeline. Other components in the pipeline, specifically Feature Extraction and ReID, require these individual detections, so the detection based approach better aligns with our goal.

Chapter 4

Design Methods

This chapter outlines our approach to solving the Corridor Counting problem. We will go over the existing solutions to similar problems that we utilize, discuss why we chose them, and then explain how our pipeline makes use of them to solve the larger problem.

4.1 Overview

As mentioned in Section 1.2, our solution to the Corridor Counting problem will make use of an existing solution to the MCVT problem. We rework and expand upon the components used by this solution to create ours. Figure 4.1 shows the full overview of dependencies of components present in our solution. As can be seen in the figure, there is significant overlap in components used in Multi-Camera Vehicle Tracking, shown in black, with the components needed to implement our solution, shown in blue.

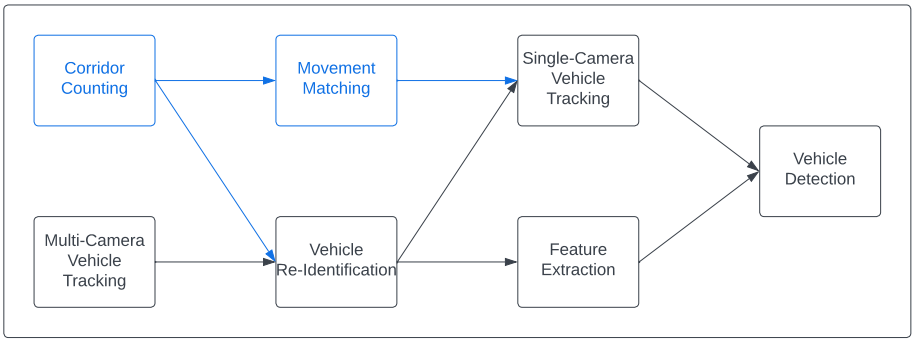

Figure 4.1: Component Dependency Diagram

We elected to use parts of the MCVT algorithm submitted by team TAG [32] in track 1 of the 2022 AI City Challenge [41]. Our decision to use their algorithm was a result of its effectiveness, for which it had an IDF1 score of 0.8371, placing it 3rd overall in the challenge, as well as its ease of use on the WAVE HPC [2], which we built and tested our pipeline on.

4.2 Components

The full pipeline for our final solution is shown in Figure 4.2. Each component is shown in black and their outputs in blue. The annotations used by the algorithm are shown in red and the input data in green. The following sections will go into detail about each component, as well as our contributions to them.

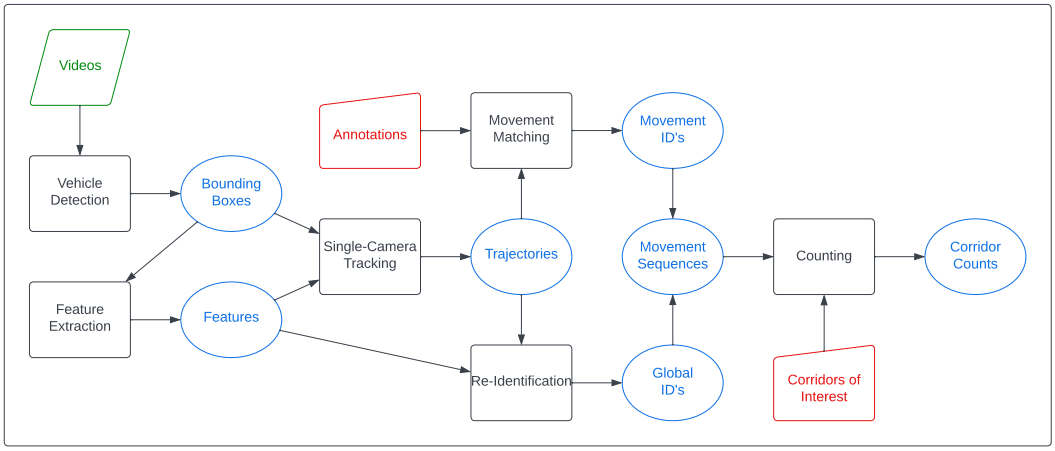

Figure 4.2: Full Pipeline Diagram

4.2.1 Vehicle Detection

Our pipeline begins by performing object detection on the target videos. This is done for every video at once before moving on to the next steps. We made use of team TAG's Vehicle Detection module, which itself is YOLOv5x [42]. This module did have the added step of first extracting all the frames from the videos and then performing detection based on those extracted images. This approach uses up a lot of storage space and could likely be removed if limited storage becomes an issue. However since our dataset was small for this project and these generated images were an intermediate step that could be deleted once Vehicle Detection finished, we did not implement this change. The end result of this step is a list of bounding boxes and a large set of images, which are cropped to the bounding boxes for each vehicle in every frame.

4.2.2 Feature Extraction

The next step is Feature Extraction, which is important for performing both Vehicle Tracking and ReID later in the pipeline. For this we used team TAG's component, including their pre-trained models. In total they have three Feature Extraction models, which are each independently used on the full set of cropped images from the Vehicle Detection step. Once they have all been run, the features are merged together to get a single representation for each instance of a vehicle.

4.2.3 Single-Camera Vehicle Tracking

Once the Detection and Feature Extraction stages are complete we begin tracking vehicles on a per camera basis. This algorithm links vehicles movements together frame by frame, resulting in a tracklet, or trajectory, that describes the path taken by a vehicle in the video. With each vehicle now being tracked between frames, it is also possible to give them local id's to identify each vehicle in a video. These id's are only unique to each video. One vehicle that appears in multiple videos will have a different local id for each video. For this stage we used team TAG's algorithm, which relied on Re-ID features and IOU to do the tracking between frames.

4.2.4 Movement Matching

At this stage in the pipeline, each vehicle is described by its local id, trajectory, and feature array. The Movement Matching step takes the vehicle trajectory and attempts to match it to one of the pre-defined movements associated with the intersection or stretch of road in the video. To do this, we were inspired by the approach presented in Ospina and Torres [12].

Each camera has its own annotations that we manually created. They consist of vectors corresponding to each movement of interest and exit lines to determine when to assign a vehicle its movement. The movement vectors are compared with a vehicle's trajectory to determine which is most similar to it. The vehicle is assigned the movement id of the most similar vector. The exit lines allow us to reduce the number of vector comparisons by ignoring movements that don't end at the specific exit. These vectors and lines can be seen in Figure 4.3. The blue lines represent the movement vectors, with the blue dot representing the head of each vector. The red lines are the exit lines. Each movement vector is associated with exactly one exit line, specifically the exit line nearest where the vector ends.

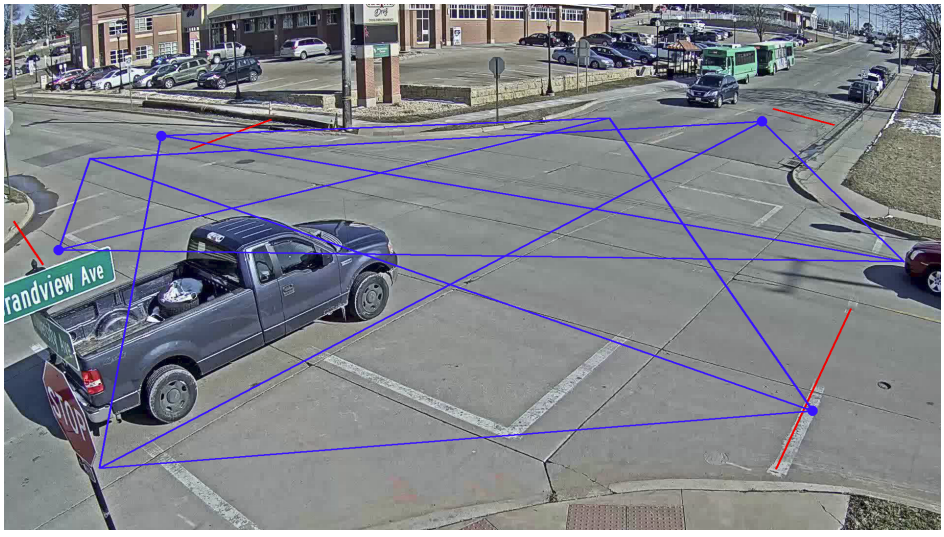

Figure 4.3: Annotated movement vectors (blue) and exit lines (red) for cam 28.

When a vehicle's bounding box crosses one of the exit lines we attempt to assign it a movement. The vehicle's trajectory is interpreted as a vector that begins at the center of the bounding box of the first frame it was detected and ends at the center of the bounding box for its current position. The movements associated with the exit the vehicle crossed are labeled as candidates. The cosine similarity between the trajectory vector and each candidate vector is calculated. The movement id given to the vehicle is the id corresponding to the candidate with the highest similarity. However, if no candidate vector had a score above a certain threshold, the vehicle was not assigned a movement at this current frame.

We found this simple approach to vehicle matching to be effective for a variety of different camera perspectives. With every camera in our dataset, we used a threshold of 0 for selecting a candidate vector. This meant that if the angle between the trajectory vector and candidate vector was 90 degrees or more, the candidate would be disregarded. This served to account for any poorly drawn exit lines or exit lines on cameras with perspectives that cause vehicles to cross multiple exit lines. Figure 4.4 shows an example of the latter. The white vehicle at the bottom crosses the left exit line before it reaches its intended exit line on the right. Moving the left line to avoid this vehicle could lead to the correct vehicles missing it. With the threshold approach, the white vehicle's trajectory is not matched with any candidates associated to the left exit and can then be matched correctly in later frames when it reaches the right exit.

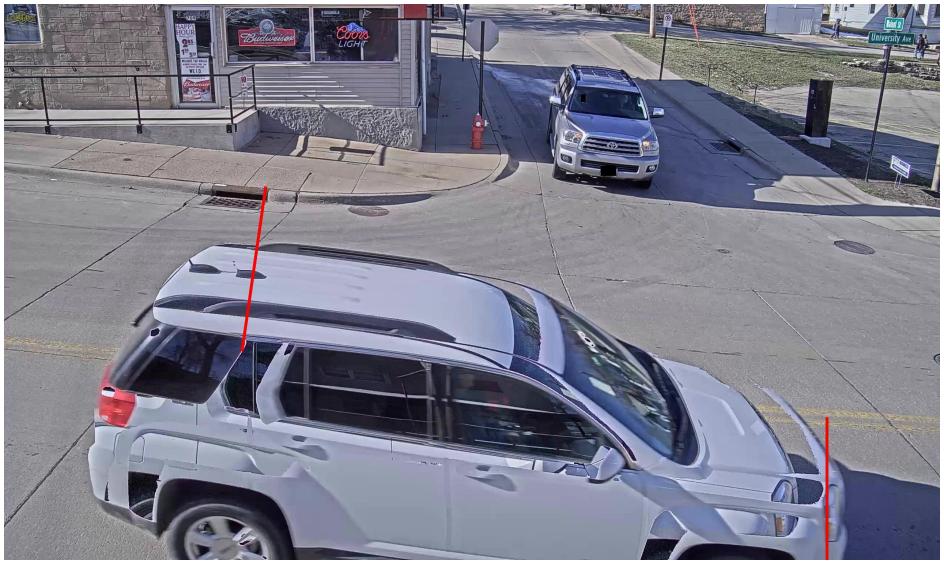

Figure 4.4: Cam 19's perspective causes a vehicle to cross an exit line in another lane.

Restricting movement assignments to just frames when a vehicle crosses an exit line has an additional benefit. Due to limitations in Vehicle Detection and Tracking, a vehicle's trajectory is sometimes split. This can be caused by occlusion from other vehicles or objects, resulting in one vehicle being treated as two. Assuming this split happens away from exit lines, only one of the two trajectories will be assigned a movement as only one will intersect an exit line. This can be seen in Figure 4.5, which shows the vectors of all unmatched vehicles. The telephone pole on the

right of the image is the cause of a number of trajectory splits, and the few on the left are likely due to the vehicles entering or exiting frame and giving the Vehicle Tracking trouble due to the limited visible features.

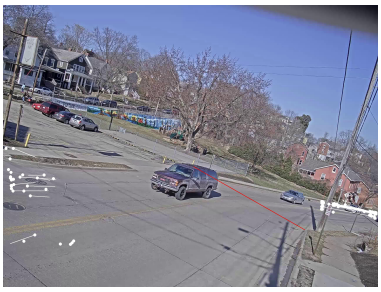

Figure 4.5: Trajectories of all unmatched vehicles from cam 18.

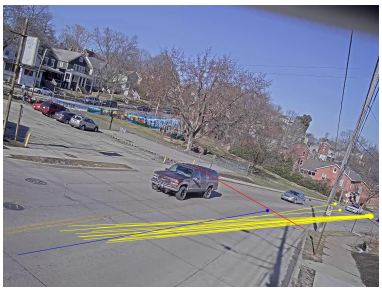

Figure 4.6: Trajectories of all vehicles matched to movement 1 from cam 18.

Figure 4.7: Trajectories of all vehicles matched to movement 2 from cam 18.

Figures 4.6 & 4.7 show the trajectory vectors of all the vehicles that were matched to their corresponding movements. Just as before, the blue lines represent the pre-defined movements each trajectory was assigned to, and the dots represent the end of the vectors.

4.2.5 Re-Identification

With each vehicle from all cameras now assigned a movement, we must link vehicles between cameras. This process makes use of the features generated in the Feature Extraction stage. For this we built upon the method used by team TAG for their Vehicle ReID component.

Their approach considered the physical features of each vehicle, as well as some spatial and temporal details regarding the relative locations of the cameras. The spatial and temporal information allowed them to cleverly divide up the ReID process. For example, it would not be ideal to compare a vehicle on *cam 41* with a vehicle on *cam 46* as it would need to go through all the other cameras first. Instead, they first compare vehicles on adjacent cameras, and then infer which vehicles are the same for non-adjacent cameras. This not only reduced the memory requirement for the ReID process by dividing it into independent tasks, but also lowers the chance of misidentifying two vehicles as the same. Additionally, they were able to adjust the probability two vehicles on different cameras were the same vehicle based on when they appeared on the cameras. All of this worked well for them because the cameras for their dataset were relatively evenly spread out along a single road, as seen in Figure 4.8 However, their implementation of this did not generalize well and would require excessive manual labeling to provide the temporal information to be used on our camera network, seen in Figure 4.9.

Due to time limitations, we opted to simplify their ReID approach for our solution. First, we removed the temporal factor due to the overhead needed to generate this information for each new camera network. From there, the next step involved dividing the vehicles into groups based on adjacent cameras. The idea behind this being if we know cameras

Figure 4.8: Camera locations of team TAG's dataset (Test set of CityFlowV2 [1]).

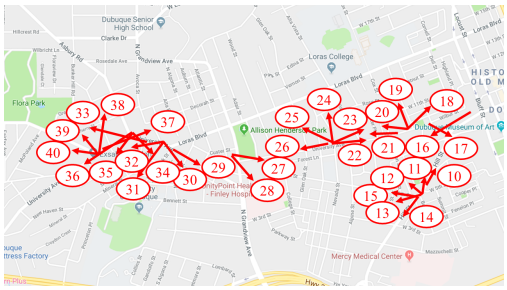

Figure 4.9: Camera locations of our dataset (Validation set of CityFlowV2 [1]).

C_i and C_j are adjacent, if a vehicle appears on C_i, it is very likely to also appear on C_j. So we create a mapping, A, to identify all adjacent cameras, where:

$$A(c_i) = \{\ c \in C \mid c \text{ is adjacent to } c_i\ \}$$

This mapping was performed manually according to the layout of the cameras presented in Figure 4.9. Then we can determine all the groups, G, vehicle v_i from camera c belongs to according to:

$$G(v_i) = \{\ (c,\ c_j) \mid c_j\ \epsilon\ A(c)\ \}$$

For this step the order of c and c_j are not important, as (c_i, c_j) and (c_j, c_i) identify the same group. At this point, each group contains a list of vehicles that appeared in the corresponding pair of cameras. Using this, we can perform clustering on the features of all vehicles in a group to re-identify cars appearing on adjacent cameras. Then we can deduce the ReID for the rest of the cameras knowing that if vehicles v_a and v_b from cameras c_i and c_j, respectively, are the same vehicle, and v_b and v_c from cameras c_j and c_k, respectively, are the same vehicle, then v_a and v_c must be the same vehicle.

With the ReID complete we can now assign a universally unique id to each vehicle. This id can be used to identify a vehicle regardless of which camera it appears on.

4.2.6 Counting

The final step is to count the number of vehicles that traveled along each of the pre-defined corridors. The counts are kept as running totals, $N_{C \times F}$, with C being the number of corridors and F the number of frames in the videos. So $N(c, f)$ is the number of vehicles that have completed corridor c after the first f frames of the videos.

Each corridor is defined as a sequence of camera and movement id pairs, S. We check each vehicle's full sequence, V to see if any of the sequences in S are uninterrupted sub-sequences of V. If this is the case for any sequence in S, it means the vehicle travelled along the full corridor described by the sequence without any detours in between. Then we can increment the values of $N(c, f)$ for $f >$ the frame the vehicle was assigned its last movement in corridor c.

Chapter 5

Evaluation

This chapter describes the techniques used for evaluating our results. We first extracted counts from the ground truth, which involved some pre-processing as the data was formatted differently. Then we performed both a manual and empirical analysis of the resulting counts.

5.1 Pre-Processing Ground Truth

The labeled data we were provided contained information about every vehicle, even those that did not complete any corridors. To get an accurate ground truth running total, we first had to filter for just the vehicles that completed at least one corridor. Additionally, the information provided for each vehicle included a universal id, a physical description, the camera it was seen on, the range of frames in which it appeared, and the movement it made. There were differences in how vehicle movements were described, using a pair of cameras titled *from_camera* and *to_camera* instead of a single movement id. So to make it easier to compare to our predictions, we converted the pairs of *from_camera*'s and *to_camera*'s to the corresponding movement id according to our annotations.

Since the data now resembled our prediction data, we were able to extract the ground truth counts. This was done by first retrieving the sequence of movements made by each vehicle according to their universal ids. Then we compared these sequences to the sequences of movements needed to complete each corridor. This resulted in a running total in the same format as the one for the prediction. The format of the ground truth counts can be seen in Table 5.1 It is these two totals that we compare to evaluate our pipeline.

Table 5.1: Ground Truth counts as a running total

Corridor	Frame 0	Frame 1	...	Frame 829	Frame 830	...	Frame 3998	Frame 3999
0	0	0	...	17	17	...	39	39
1	0	0	...	21	21	...	47	47
2	0	0	...	6	7	...	23	23
3	0	0	...	25	25	...	79	79

5.2 Visual Analysis

We will first compare the final counts of each corridor from the prediction and ground truth manually. These totals can be seen in Table 5.2. The predictions for the first two corridors are significantly lower than the true counts, meanwhile the last two corridors overshoot the true counts by only a few vehicles.

For corridor 0 this is likely due to how many movements define the corridor. While the other three corridors are only 3 movements, corridor 0 is defined by 8 movements. With so many more cameras, ReID needs to be performed more. This provides additional opportunities for the ReID to not link two occurrences together, causing the vehicle's sequence of movements to be split and not satisfy the corridor.

Table 5.2: Final counts of Ground Truth and Predictions for each corridor

Corridor	Ground Truth	Prediction
0	39	29
1	47	38
2	23	26
3	79	81

However for corridor 1, which has a similarly sized under prediction, this is not the case. For this corridor it is likely due to something on *cam 26*. Corridor 2, which had a very small over prediction, is almost the same set of movements as corridor 1. The only difference being it contains a movement from *cam 27* instead of from *cam 26*. So we can conclude that something occurs on *cam 26* that causes significant error for corridor 1 in comparison to *cam 27* for corridor 2. Looking at *cam 26* and the assigned movements for it shown in Figures 5.1 & 5.2, we can see there is nothing particularly challenging about this perspective, nor are there any glaring issues with the assigned movements for each vehicle trajectory. Next steps for finding the cause of the significant under prediction would be manually verifying the ReID for vehicles traversing through corridor 1, as well as verifying the ground truth labels. However, for now we are not sure what the cause of this is.

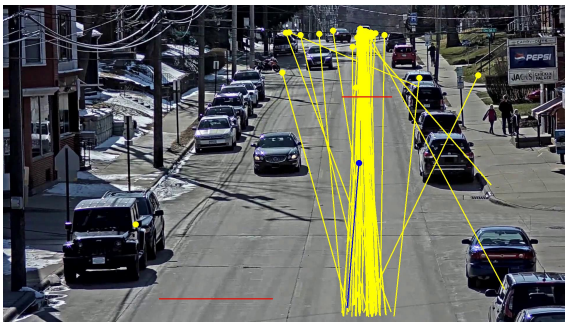

Figure 5.1: Trajectories of all vehicles matched to movement 1 from cam 26.

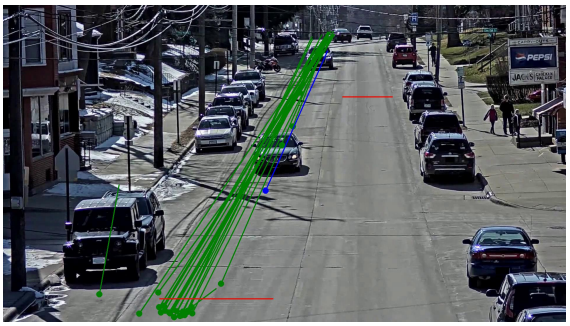

Figure 5.2: Trajectories of all vehicles matched to movement 2 from cam 26.

Corridors 2 and 3 overestimating their ground truths is possibly a result of the ReID incorrectly linking two vehicles together, thus combining their sequences and completing the corridor. We can see this occurring in Figures 5.3 & 5.4, where a white Lincoln SUV is linked to a white police SUV. As with corridor 1, further analysis of the ReID results could give us insight into the incorrect predictions.

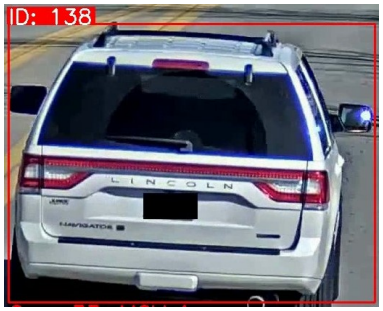

Figure 5.3: A white Lincoln SUV on cam 29 given ID 138.

Figure 5.4: A white Ford police car on cam 27 given ID 138.

5.3 Quantitative Analysis

To gain further insight into our algorithm, we will use a metric designed to compare predicted and ground truth running totals. This method is taken from the AI City 2020 Track 1 efficiency score evaluation algorithm for Single-Camera Vehicle Counting [20]. Although this algorithm was designed to evaluate vehicle counts for a single camera, it can be applied to our corridor counts since they are formatted in the same running total format.

5.3.1 Evaluation Methodology

The metric used is a segmented, weighted root mean square error (wRMSE). For this, each corridor's running total is divided into k segments, where $1 \leq k \leq max(\text{video frame counts})$. The last frame in each segment is used to calculate an RMSE. This segmentation approach is to reduce the error caused by labeling inconsistencies resulting in variations of the frame in which a vehicle is counted. A smaller value of k is more forgiving to inaccurate intermediate counts, as there are fewer segments near the beginning of the video where discrepancies in the count are more likely. The segment RMSE's are then weighed to further favor later segments, when the discrepancies in the running total are more likely to be resolved. This metric is calculated by the following equations:

$$wRMSE = \sqrt{\sum_{i=1}^{k} w_i \left(\hat{x}_i - x_i\right)^2}$$

$$w_i = \frac{i}{\sum_{j=1}^{k} j} = \frac{2i}{k\left(k+1\right)}$$

The wRMSE score describes how closely the predicted count matches the ground truth count. However this score tends to be larger when the total count of vehicles completing a corridor is high. To fix this they proposed a normalized weighted root mean square error (nwRMSE) according to the following:

$$nwRMSE = \begin{cases} 1 - \dfrac{wRMSE}{\text{true vehicle count}} & \text{if } wRMSE \leq \text{true vehicle count} \\ 0 & \text{if } wRMSE > \text{true vehicle count} \end{cases}$$

We will look at both the wRMSE and nwRMSE scores for each corridor to get an idea of how well our algorithm performed.

5.3.2 Results

To gain further insight into the performance of our algorithm, we examine the wRMSE and nwRMSE. We decided to calculate the scores using $k = 50, 100, 500, 1000$. This will allow us to see how the scores are affected by evaluating them with more segments and hopefully get more information about the intermediate counts. Table 5.3 shows the scores we calculated for the specified values of k.

Table 5.3: RMSE values of each corridor with various k values

	$k = 50$		$k = 100$		$k = 500$		$k = 1000$	
Corridor	wRMSE	nwRMSE	wRMSE	nwRMSE	wRMSE	nwRMSE	wRMSE	nwRMSE
0	7.168	0.816	7.026	0.820	6.626	0.830	6.623	0.830
1	7.657	0.837	7.709	0.836	7.160	0.848	7.161	0.848
2	1.790	0.922	1.638	0.929	1.762	0.923	1.742	0.924
3	4.170	0.947	4.330	0.945	4.609	0.942	4.615	0.942

The nwRMSE scores show a small increase for corridors 0, 1, and 2 as the number of segments, k, increases. This implies that the earlier counts for these corridors are not significantly far off when compared with the counts towards the end of the videos. Corridor 3 is the exception, where the nwRMSE decreases with the growing values of k, implying there were discrepancies in the count early on that improved towards the later segments. However, the intermediate counts themselves are not too important a metric to examine, as their accuracy depends on which frames the person labeling decided to count each vehicle. Ultimately, it seems the discrepancies in each corridor's predicted counts come down to mistakes in the ReID stage.

Chapter 6

Societal Issues

This chapter examines the societal impact of our project. It is important to consider the potential misuse of our Corridor Counting algorithm, the broader societal implications, and the steps that can be taken to mitigate any potential harm.

6.1 Ethical Concerns

The major concern with our project is the potential for privacy violations. Given access to multiple videos, our algorithm can be used for surveillance. While its main purpose is to count vehicles in a corridor, it must first identify where individual vehicles travel, making it easy to create a modification for tracking specific vehicles. For instance, the technology could be used to search for specific vehicles and others like it, and then find where these vehicles went. Care must be taken to prevent abuse. Ultimately, protecting privacy rights against such surveillance requires legislation, as it is nearly impossible for our algorithm and others like it to achieve its goal without opening the door to this potential misuse.

6.2 Environmental Impact

The environmental impact of our work is twofold. We must consider both the effects of running our algorithm and the events that unfold as a result of the data it makes available.

The first aspect involves the energy demand of our algorithm. The full pipeline of our solution is computationally complex, requiring around 13 hours to run when utilizing 2 NVIDIA Tesla V100 GPUs [43]. With each of these GPUs drawing 250W, the overall power consumption of our pipeline can be quite high. If this was expanded to a larger dataset, with either more cameras in the network or longer videos, there should be an effort made to reduce the required energy.

On top of energy demand, our algorithm provides data that can contribute to more efficient road networks. Having information about traffic patterns can allow city planners to create and alter roads in a manner that reduces overall commute time. This improvement would result in fewer emissions from vehicles as they spend less time on the road.

6.3 Usability

One of the minor goals for this project was to improve the ease of use of the pipeline in comparison to the base MCVT algorithm. To this end, we made efforts to organize and simplify the directory structure and to format the manual annotations to be easily interpretable. Additionally, we created and altered a number of scripts to enable the full pipeline to be run with a single command. All the necessary setup information and commands to use the algorithm is provided in the documentation. So, while creating the initial annotations for a camera network is inevitably time consuming, the process of providing them to and running the algorithm is very user friendly.

Although we did our best to increase the user-friendliness of our pipeline, it is still a very complex code base. As such, there is a big learning curve for understanding all the components and the intermediate data the algorithm produces. This means it can be a difficult and lengthy process to make improvements to our solution.

Chapter 7

Constraints and Standards

The direction of our project was shaped by the input data we had access to and the desired format for the output. This chapter goes over all the constraints we faced when designing our algorithm as well as the standards we followed.

7.1 Design Constraints

The dataset we were provided for this project was CityFlowV2 [1]. This dataset consisted of 46 videos taken from numerous cameras located around a mid-sized US city. The videos were of mixed resolution but overall high quality, contained a variety of angles and perspectives, and had a frame rate of 10 frames per second. This meant we had to consider cases where vehicles are occluded by other vehicles or move large distances between frames when designing certain phases of our algorithm.

Since we were able to build upon existing solutions to the MCVT problem, we had to select which solution to use. For ease of use, the options we considered were those submitted to the 2022 AI City Challenge [41]. These solutions were all implemented on the CityFlowV2 dataset, making it easier to implement our own solution which also used this dataset. We tested many of the top performing solutions, some of which were not compatible with our system, the WAVE HPC, due to their usage of technologies such as Docker [3] or either deprecated or conflicting libraries. Our choice of solution was then limited to these options.

The CityFlowV2 dataset is divided into three groups: train, test, and validation. In order to evaluate the performance of our final solution, we were provided labels for 16 videos in the validation set. These labels described each vehicle seen across all the videos, including which movement they made and which corridor they were traversing. These labels determined which videos we would use on our algorithm, being the ones in the validation set, as well as which corridors we would count vehicles across. While this did not affect the algorithm itself, it did mean we would have to annotate these videos for the Movement Matching phase as discussed in Section 4.2.4. Additionally, we had to make changes to the MCVT solution we used since all these solutions were implemented on the test set. There ended up being many annotations required for each camera, however due to the time limit of the project being the end of the

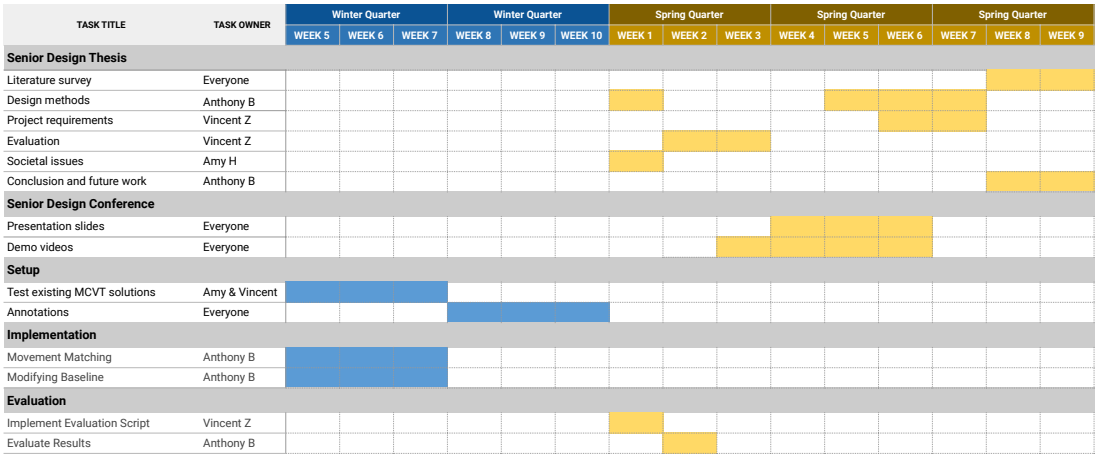

Figure 7.1: Gantt Timeline showing Project Progress

year, we had to make some adjustments to the Re-Identification phase described in Section 4.2.5 to reduce the number of annotations we needed. The final timeline of our work can then be seen in Figure 7.1.

7.2 Standards

As discussed in Section 2.1, the input videos are from CityFlowV2. When Vehicle Detection is performed on these videos, the frames are first extracted from the original .avi files into individual .jpg files, as specified by ISO/IEC 10918-1 [44]. As such, this phase uses OpenCV [45] due to its support for a large variety of image and video encodings. These image files are further used throughout the pipeline, so we utilize OpenCV in any instance where data needs to be processed from the .jpg files.

There are a number of annotations required for the Movement Matching and Counting phases of our pipeline. We elected to organize these into JSON files, specified in RFC 8259 [46] and ECMA-404 [47], to be both easily understandable by humans for future annotations and simple to parse for our algorithm. Furthermore, this arrangement will allow for future simplifications to the annotation process that will be discussed in Section 8.2.2.

We mentioned concerns with misuse of this algorithm in Section 6.1 that would violate the privacy of those in the videos used. One step that can be taken to address this issue is establishing a Privacy Information Management System as described in ISO/IEC 27701 [48]. This standard provides guidelines for identifying and managing the privacy risks that come from processing data, such as the videos that are run through our pipeline which contain license plates and faces of drivers and pedestrians. It describes the steps we can take towards reducing privacy risks and comply with relevant laws and regulations.

Chapter 8

Conclusion and Future Work

This chapter will summarize the work we completed for the Corridor Counting algorithm. We will discuss areas in which we fell short, as well as what could be done going forward to further improve the performance of the algorithm.

8.1 What We Accomplished

As we built upon the MCVT solution presented by team TAG, our work was focused on improving their algorithm and expanding upon it to solve the Corridor Counting problem. In doing so, we implemented a Movement Matching algorithm inspired by Ospina and Torres [12] to establish a sequence of movements performed by each vehicle. We introduced the idea of movements corresponding to an exit to improve performance on obscure camera perspectives. After this we altered team TAG's ReID component to better generalize for other camera networks, since our network was much more complex than what their algorithm was designed for. With this done, we implemented a sequence comparison algorithm to determine if vehicles completed a corridor then add it to the count for the final output.

8.2 Future Work

We have identified two key areas where improvements to the effectiveness and ease of use of our Corridor Counting algorithm can be made. The first is in the ReID phase to improve the overall performance of this component. The second is for simplifying the process of creating annotations for each camera to be used in the Movement Matching phase.

8.2.1 Re-Identification

As discussed in Section 5.2, ReID appears to be a consistent cause of error across all corridors we had. In Section 4.2.5 we described how we simplified this step from the one used by team TAG. While the temporal information we did not include could be useful for improving performance, we still deem it as being too large an overhead to incorporate it in the future.

An alternative is to improve how we divide vehicles into groups to perform ReID. Currently we divide vehicles based on camera adjacency. However, with the addition of the Movement Matching component, we now have information about where each vehicle is heading once it leaves a camera's perspective. The resulting camera, $R(c, m)$, is dependent on the camera the vehicle appears on, c, and the movement it makes, m. This information allows us to assign a vehicle to a group that corresponds only to the camera it was seen on and the camera it is heading to. More formally, each vehicle, v_i from camera c, is assigned one group, G, according to:

$$G(v_i) = (c, R(c, m))$$

As with the previous approach for assigning groups, the order of cameras does not matter, meaning (c_i, c_j) and (c_j, c_i) represent the same group.

The benefit of forming groups in this manner is that they can be derived from a graph representation of the camera network. If we construct a directed graph where each camera is represented by a node and each possible movement is an outgoing edge given by $(c, R(c, m))$ with name m, then it is easy to derive the group for each vehicle according to:

$$G(v_i) = (c, c- > m)$$

where $c- > m$ denotes the camera node reached from following edge m from node c.

Since determining the resulting camera of each movement requires knowledge of the full camera layout, this is something that has to be done manually. We did not have time to do this so we were unable to incorporate this approach into our solution. However, the next area of improvement could make this process much easier.

8.2.2 Annotations

Current efforts to annotate movement vectors and exit lines for each camera are quite inefficient. It involves using a photo editing program to find the desired coordinates, then entering them into the lists of vectors and lines. Then there is an additional step of entering the corresponding exit for each movement vector.

The improvement we had in mind entails creating a program that allows the user to draw the movement vectors and exit lines directly onto an image. This would make applying our algorithm on new camera networks easier, as it would simplify the steps for annotation and automate some parts of it.

The annotation program would iterate through a still of each camera in the data set. On each still, the user would have the option to play the video to get an idea of how vehicles traverse the intersection or street. From there, they can draw all the necessary exit lines, which are automatically assigned ids. Once those are drawn they can begin drawing movement vectors. For each vector the user will be asked to enter the id of the corresponding exit line, and if known, the camera id that a vehicle taking said movement is heading towards. With this extra information, the program will

then be able to format all the annotations to be used by our algorithm and can even generate the graph of the camera network automatically.

This program would make annotating cameras much simpler, but it also speeds the whole process up and reduces the chance for errors that can have a significant affect on the performance of our algorithm.

8.3 Why It Is Important

The information about traffic patterns that can be learned from the data our algorithm provides is invaluable to city planners and others involved in managing traffic. Our algorithm can be applied on many different networks of cameras to gain insight into how people traverse between two areas of a city, which regions are most commonly used, and how traffic flows through stretches of road. This information can then lead to improvements in signal timing, congestion mitigation strategies, and future road designs, all ultimately resulting in safer and more efficient roads for both drivers and pedestrians.

Bibliography

[1] Z. Tang, M. Naphade, M.-Y. Liu, X. Yang, S. Birchfield, S. Wang, R. Kumar, D. Anastasiu, and J.-N. Hwang, "Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification," 2019.

[2] Santa Clara University, *WAVE High Performance Computing (HPC)*, 2023. https://wiki.wave.scu.edu/doku.php?id=wave_hpc.

[3] "Docker: Empowering app development for developers." https://www.docker.com.

[4] A. B. Yoo, M. A. Jette, and M. Grondona, *Slurm Workload Manager*, 2003. https://slurm.schedmd.com/.

[5] K.-H. N. Bui, H. Yi, and J. Cho, "A vehicle counts by class framework using distinguished regions tracking at multiple intersections," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2466–2474, 2020a.

[6] S. Bose, C. D. Ramesh, and M. H. Kolekar, "Vehicle classification and counting for traffic video monitoring using yolo-v3," in *2022 International Conference on Connected Systems & Intelligence (CSI)*, pp. 1–8, 2022.

[7] K.-H. N. Bui, H. Yi, and J. Cho, "A multi-class multi-movement vehicle counting framework for traffic analysis in complex areas using cctv systems," *Energies*, vol. 13, no. 8, 2020b.

[8] X.-D. Nguyen, A.-K. N. Vu, T.-D. Nguyen, N. Phan, B.-D. D. Dinh, N.-D. Nguyen, T. V. Nguyen, V.-T. Nguyen, and D.-D. Le, "Adaptive multi-vehicle motion counting," *Signal, Image and Video Processing*, vol. 16, no. 8, pp. 2193–2201, 2022.

[9] K. Kollek, M. Braun, J.-H. Meusener, and A. Kummert, "Real-time traffic counting on resource constrained embedded systems," in *2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1–4, 2022.

[10] H. Lin, Z. Yuan, B. He, X. Kuai, X. Li, and R. Guo, "A deep learning framework for video-based vehicle counting," *Frontiers in Physics*, vol. 10, p. 829734, 2022.

[11] Z. Liu, W. Zhang, X. Gao, H. Meng, X. Tan, X. Zhu, Z. Xue, X. Ye, H. Zhang, S. Wen, and E. Ding, "Robust movement-specific vehicle counting at crowded intersections," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2617–2625, 2020.

[12] A. OSPINA and F. TORRES, "Countor: count without bells and whistles," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2559–2565, 2020.

[13] Z. Wang, B. Bai, Y. Xie, T. Xing, B. Zhong, Q. Zhou, Y. Meng, B. Xu, Z. Song, P. Xu, R. Hu, and H. Chai, "Robust and fast vehicle turn-counts at intersections via an integrated solution from detection, tracking and trajectory modeling," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2598–2606, 2020.

[14] 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.

[15] 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.

[16] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[17] T.-H. Chen, Y.-F. Lin, and T.-Y. Chen, "Intelligent vehicle counting method based on blob analysis in traffic surveillance," in *Second International Conference on Innovative Computing, Informatio and Control (ICICIC 2007)*, pp. 238–238, 2007.

[18] Y. Chen and J. Lu, "A video-based method for vehicle counting based on motion and color features," in *2021 9th International Conference on Traffic and Logistic Engineering (ICTLE)*, pp. 41–45, 2021.

[19] L. Janos Lance, T. Edwin Sybingco, and J. A. C. Jose, "Efficient vehicle counting algorithm using gaussian mixing models and iou based tracker for lpwan based intelligent traffic management systems," in *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, pp. 1–4, 2022.

[20] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, X. Yang, L. Zheng, A. Sharma, R. Chellappa, and P. Chakraborty, "The 4th ai city challenge," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2665–2674, 2020.

[21] J. Zhang, J. Chen, J. Cao, R. Liu, L. Bian, and S. Chen, "Dual attention granularity network for vehicle re-identification," *Neural Computing and Applications*, vol. 34, pp. 2953–2964, 02 2022.

[22] X. Guo, J. Yang, X. Jia, C. Zang, Y. Xu, and Z. Chen, "A novel dual-pooling attention module for uav vehicle re-identification," 2023.

[23] G. Jiang, X. Pang, X. Tian, Y. Zheng, and Q. Meng, "Global reference attention network for vehicle re-identification," *Applied Intelligence*, vol. 53, pp. 1–16, 09 2022.

[24] W. Zhu, Z. Wang, X. Wang, R. Hu, H. Liu, C. Liu, C. Wang, and D. Li, "A dual self-attention mechanism for vehicle re-identification," *Pattern Recognition*, vol. 137, p. 109258, 2023.

[25] F. Ahmad, M. Z. Ansari, S. Hamid, and M. Saad, "A computer vision based vehicle counting and speed detection system," in *2023 International Conference on Recent Advances in Electrical, Electronics & Digital Healthcare Technologies (REEDCON)*, pp. 487–492, 2023.

[26] Z. Al-Ariny, M. A. Abdelwahab, M. Fakhry, and E.-S. Hasaneen, "An efficient vehicle counting method using mask r-cnn," in *2020 International Conference on Innovative Trends in Communication and Computer Engineering (ITCE)*, pp. 232–237, 2020.

[27] P. Bergmann, T. Meinhardt, and L. Leal-Taixé, "Tracking without bells and whistles," *CoRR*, vol. abs/1903.05625, 2019.

[28] Y. Zhao, "Design and implementation of vehicle tracking system based on depth learning," in *Proceedings of the 2nd International Conference on Advances in Image Processing*, ICAIP '18, (New York, NY, USA), p. 138–143, Association for Computing Machinery, 2018.

[29] L. Yu, Q. Feng, Y. Qian, W. Liu, and A. G. Hauptmann, "Zero-virus: Zero-shot vehicle route understanding system for intelligent transportation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2534–2543, 2020.

[30] T. Xu, K. Shen, Y. Fu, H. Shi, and F. X. Lin, "Rev: A video engine for object re-identification at the city scale," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pp. 189–202, 2022.

[31] Q. Wang, Y. Zhong, W. Min, H. Zhao, D. Gai, and Q. Han, "Dual similarity pre-training and domain difference encouragement learning for vehicle re-identification in the wild," *Pattern Recognition*, vol. 139, p. 109513, 2023.

[32] H. Yao, Z. Duan, Z. Xie, J. Chen, X. Wu, D. Xu, and Y. Gao, "City-scale multi-camera vehicle tracking based on space-time-appearance features," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3309–3317, 2022.

[33] B. A. Holla, M. M. M. Pai, U. Verma, and R. M. Pai, "Enhanced vehicle re-identification for smart city applications using zone specific surveillance," *IEEE Access*, vol. 11, pp. 29234–29249, 2023.

[34] J. Zhang, J.-J. Qiao, X. Wu, and W. Li, "Vehicle counting network with attention-based mask refinement and spatial-awareness block loss," in *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21, (New York, NY, USA), p. 2889–2898, Association for Computing Machinery, 2021.

[35] S. Zhang, G. Wu, J. P. Costeira, and J. M. F. Moura, "Fcn-rlstm: Deep spatio-temporal neural networks for vehicle counting in city cameras," 2017.

[36] C.-J. Lin, S.-Y. Jeng, and H.-W. Lioa, "A real-time vehicle counting, speed estimation, and classification system based on virtual detection zone and yolo," *Mathematical Problems in Engineering*, vol. 2021, p. 1577614, 11 2021.

[37] B. Hardjono, M. G. A. Rhizma, A. E. Widjaja, H. Tjahyadi, and M. J. Josodipuro, "Vehicle counting evaluation on low-resolution images using software tools," in *ICICM 2019: Proceedings of the 9th International Conference on Information Communication and Management*, ICICM 2019, (New York, NY, USA), p. 89–94, Association for Computing Machinery, 2019.

[38] C.-M. Tsai, F. Y. Shih, and J.-W. Hsieh, "Real-time vehicle counting by deep-learning networks," in *2022 International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 175–181, 2022.

[39] R. Kejriwal, R. H J, A. Arora, and Mohana, "Vehicle detection and counting using deep learning basedyolo and deep sort algorithm for urban traffic management system," in *2022 First International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, pp. 1–6, 2022.

[40] J. Mirthubashini and V. Santhi, "Video based vehicle counting using deep learning algorithms," in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 142–147, 2020.

[41] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, Y. Yao, L. Zheng, M. Shaiqur Rahman, A. Venkatachalapathy, A. Sharma, Q. Feng, V. Ablavsky, S. Sclaroff, P. Chakraborty, A. Li, S. Li, and R. Chellappa, "The 6th ai city challenge," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3346–3355, 2022.

[42] G. Jocher, *Ultralytics YOLOv5*, 2020. https://github.com/ultralytics/yolov5.

[43] NVIDIA, "Nvidia tesla v100 32g," 2018. https://www.nvidia.com/en-us/data-center/tesla-v100/.

[44] "Information technology — digital compression and coding of continuous-tone still images — requirements and guidelines," Tech. Rep. ISO/IEC 10918-1:1994, International Organization for Standardization, 1994.

[45] "Opencv: Open source computer vision library." https://opencv.org/.

[46] T. Bray, "The javascript object notation (json) data interchange format," Tech. Rep. RFC 8259, Internet Engineering Task Force (IETF), 2017.

[47] "Ecma-404: The json data interchange format," Tech. Rep. ECMA 404, Ecma International, 2013.

[48] "Security techniques – extension to iso/iec 27001 and iso/iec 27002 for privacy information management – requirements and guidelines," Tech. Rep. ISO/IEC 27701, International Organization for Standardization (ISO), 2019.