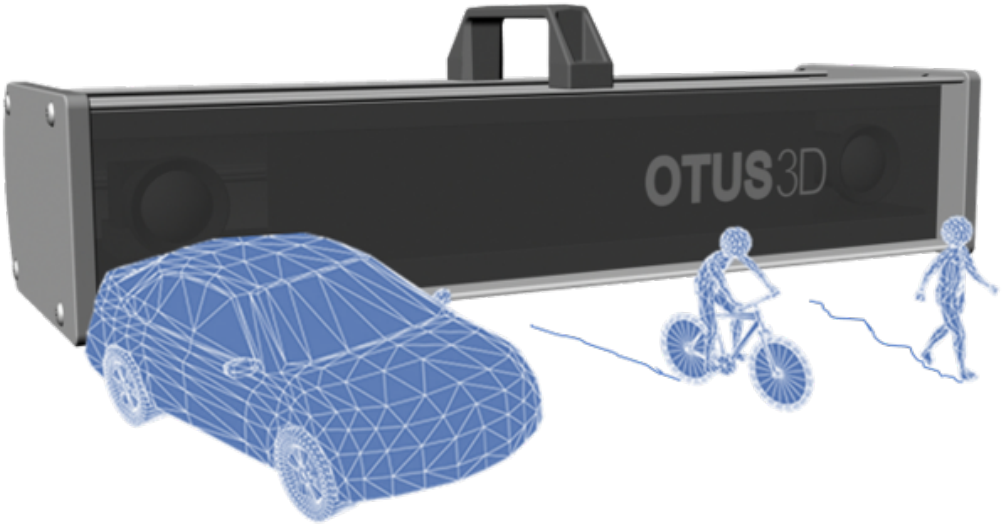
CHALMERS
UNIVERSITY OF TECHNOLOGY



Matching Traffic Objects recorded by Stereoscopic Cameras

Alignment of cameras with partially overlapping views, and matching partial trajectories

Complex Adaptive Systems

Tommy Phung
Stefan Areback

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Matching Traffic Objects recorded by Stereoscopic Cameras

Alignment of cameras with partially overlapping views, and matching partial trajectories

TOMMY PHUNG & STEFAN AREBACK

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Matching Traffic Objects recorded by Stereoscopic Cameras Alignment of cameras with partially overlapping views, and matching partial trajectories
TOMMY PHUNG & STEFAN AREBACK

Supervisor: Ulf Erlandsson, Viscando AB
Examiner: Giovanni Volpe, Department of Physics

Master's Thesis 2020
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Otus 3D camera detecting different traffic objects.

Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Matching Traffic Objects recorded by Stereoscopic Cameras
Alignment of cameras with partially overlapping views, and matching partial trajectories
Tommy Phung & Stefan Areback
Complex Adaptive Systems
Chalmers University of Technology

Abstract

If one uses several cameras to film different but overlapping parts of a scene (in our case an intersection), then a way to get an overview of the scene, is to relate all the cameras to a single coordinate system. This can be done manually using knowledge of the positions relative to the different cameras of objects that show up in the overlapping part of the filmed scene. However, it would be preferable (to save time, amongst other things) if this process could be automated, using information recorded by the cameras (in this case the positions, velocities and timestamps of traffic objects filmed by the camera). The suggested methods for achieving this is Coherent point drift (CPD) with the use of Expectation maximization (EM).

Once a common coordinate system has been found, one still needs to merge the trajectories from the different cameras corresponding to the same traffic object into a single trajectory. Preferably, this too should be done using only the information recorded by the camera (positions, velocities, timestamps). For finding a merge, the presented method is a modified version of Longest Common Subsequence (LCSS) with respect to the camera views and their overlap, presented as polygons.

CPD performs well for two cameras when LCSS is being applied as a method for noise reduction whereas when there are three cameras it gives an ambivalent solution. Using LCSS when matching trajectories for merging performs well for both two and three cameras, however the merging methods needs some additional calibrations.

Keywords: Point set registration, Coherent point drift, Expectation maximization, Longest Common Subsequence, camera alignment

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Tracking traffic objects such as cars, heavy vehicles, cyclist or pedestrians in complex traffic situations where additional cameras are involved to capture the magnitude of its complexity, there is more room for error in the form of mix ups within each group of traffic object. To track traffic objects stereoscopic cameras are used. The cameras work independently, so tracks from different cameras need to be aligned and merged into a global coordinate system. Hence, there exists a transformation problem. The transformations between the cameras coordinate systems are currently found by hand, meaning that a person is watching the tapes and correctly identifies the appropriate traffic objects between each camera to form a transformation matrix from those observation. One would thus conserve time if this process could be (semi-)automated.

The approach to find a correct transformation(s) between two cameras is to find their overlap and the points in which they share and apply some transformation methods upon those points. Some previous attempts has been made in the form of homogrophy estimation in combination with least median of squares algorithm [1] which involves choosing trajectories randomly. This method however is not robust enough against noise and has a high time complexity since it is too reliant on randomness when choosing trajectories in the data set for finding a proper transformation.

Others have used the vision of the cameras as input and using a iterative calculated probability distribution for each point whether they belong in the overlap or not [2]. The method is called Expected Overlap Estimation and is capable of finding the correct transformation if the level of noise is near nonexistent.

Where the previous sections involves time efficiency when finding a proper transformation between cameras, the second part consists of improving the data through merging. When there are multiple cameras, there is a possibility for them to register a trajectory incorrectly as two seperate ones. A way to improve the data is thus to detect the incorrect ones and merge them as one. An improved quality of data in the form of traffic is mainly in the interest of developing of autonomous cars but is also in interest overall when it comes to training neural networks. It is also of interest for cities in order to improve traffic flow and identify sources of conflicts between road users.

Previous work has been abled to produce such robust algorithms with the inclusion of colors of the traffic objects [3]. This paper seeks to find such robust methods

without the use of image analysis i.e filtered track data such as coordinates, velocity and time.

1.2 Aim

The first part of the aim is to find methods to (partially) automate the estimation of parameters for the transformations between the coordinate systems of the different cameras. The main aim of the second part is to produce a method to find which trajectories have been split and labeled as seperate objects within the different cameras.

1.3 Limitations

This paper does not aim to produce a complete merging algorithm after the split trajectories have been found. Nonetheless, a temporary merging algorithm is still being produced. More details about the temporary merging is in the discussion about matching trajectories.

2
Theory

2.1 Camera alignment

Consider two (or more) stereo cameras set up at, for example, an intersection, observing different (but overlapping) parts of the intersection. Each camera records the positions relative to the camera and velocities of traffic objects that enter its' field of vision, along with the times they were recorded. Our goal is to use these recorded positions, velocities and timestamps to find the rigid transformation relating the coordinate system of one camera to another. The method we will use to do this makes use of the Expectation maximization (EM) algorithm, which we present first.

2.1.1 EM algorithm

Suppose we have a sequence of observed random variables $X_1, \ldots, X_n$ and a sequence $Z_1, \ldots, Z_n$ of unobserved random variables generated from some statistical model depending on some parameters Θ. For our purposes the statistical model is a GMM and the unobserved random variable Z_k specifies which mixture component the random variable X_k is generated from. We want to find the maximum likelihood estimates of the parameters with respect to the observed data $X_1 = x_1, \ldots, X_n = x_n$. This means we want to find the maximum likelihood estimates of the parameters by optimizing the incomplete likelihood function $L(\theta; X)$ (or equivalently, by optimizing the incomplete log-likelihood function $\ell(\theta; X) = \log L(\theta; X)$). In the EM algorithm we will also make use of the complete log-likelihood function $\ell(\theta; X, Z)$.

In the case of a mixture model, the incomplete (or marginal) likelihod function is given by

$$L(\theta; X) = \prod_{n=1}^{N} \sum_{m=1}^{M} \pi_m \varphi_m(x_n|\theta),$$

and the complete likelihood function is given by

$$L(\theta; X, Z) = \prod_{n=1}^{N} \prod_{m=1}^{M} \varphi_m(x_n|\theta)^{\chi(Z_n=m)}.$$

The incomplete log-likelihood is in this case given by

$$\ell(\theta; X) = \sum_{n=1}^{N} \log\Big(\sum_{m=1}^{M} \pi_m \varphi_m(x_n|\theta)\Big),$$

which is usually not an easy expression to deal with. On the other hand, the complete log-likelihood function is given by

$$\ell(\theta; X, Z) = \sum_{n=1}^{N} \sum_{m=1}^{M} \chi_{(Z_n=m)} \log \varphi_m(x_n|\theta), \tag{2.1}$$

which is usually easier to deal with

In the EM algorithm, we attempt to find the maximum likelihood esimates of the parameters of the incomplete likelihood function, by an iterative procedure. In each iteration of the EM algorithm there are two steps. In the first step, called the expectation step (E step), the expected value of the complete likelihood function, with respect to the distribution of Z conditional on X and $\theta^{(r)}$, is computed. The result is a function Q of the parameter θ:

$$Q(\theta, \theta^{(r)}) = \mathbb{E}_{Z|X,\theta^{(r)}}\Big(\ell(\theta; X, Z)\Big).$$

In the second step, the maximization step (M step), the next iterate $\theta^{(r+1)}$ is found by maximizing the Q-function:

$$\theta^{(r+1)} = argmax_\theta Q(\theta, \theta^{(r)}).$$

One can show that the incomplete log-likelihood function increases as Q increases. More specifically one can show that [4]

$$\ell(X|\theta) - \ell(X|\theta^{(r)}) \geq Q(\theta, \theta^{(r)}) - Q(\theta^{(r)}, \theta^{(r)}),$$

so that an increase in Q implies at least an equal increase in the incomplete log-likelihood.

Let us illustrate the E step in the special case of a mixture model. In the case of a mixture model, the complete log-likelihood function is given by 2.1, so the Q function is given by

$$\begin{aligned} Q(\theta, \theta^{(r)}) &= \sum_n \sum_m \mathbb{E}_{Z|X,\theta^{(r)}}\Big(\chi_{(Z_n=m)}\Big) \log \varphi_m(x_n|\theta) \\ &= \sum_n \sum_m P(Z_n = m|X, \theta^{(r)}) \log \varphi_m(x_n|\theta) \end{aligned}$$

We see that in the case of a mixture model the E step amounts to computing the conditional membership probabilities $P(Z_n = m|X_1 = x_1, \ldots, X_n = x_n, \theta)$. Using Bayes' law, these can be expressed in terms of the membership probabilities π_j and the densities φ_j of the mixture components, both of which are known quantities:

$$\begin{aligned} P(Z_n = m|X, \theta^{(r)}) &= \frac{P(X_n = x_n|Z_n = m, \theta^{(r)})P(Z_n = m|\theta^{(r)})}{\sum_j P(X_n = x_n|Z_n = j, \theta^{(r)})P(Z_n = j|\theta^{(r)})} \\ &= \frac{\pi_m \varphi_m(x_n|\theta^{(r)})}{\sum_j \pi_j \varphi_j(x_n|\theta^{(r)})}. \end{aligned}$$

2.1.2 Coherent point drift

Suppose $\mathcal{X} = \{x_1, \dots, x_N\}$ and $\mathcal{Y} = \{y_1, \dots, y_M\}$ are two sets of points, and that we want to find the rigid transformation $y \mapsto Ry + v$ (i.e. R is a rotation matrix and v is a vector) that, in some sense, best aligns the set $\mathcal{Y}$ with $\mathcal{X}$. In the Coherent point drift (CPD) algorithm the idea is to consider the points $\mathcal{Y}$ to be the centroids in a mixture model, and the points $\mathcal{X}$ to be generated from this mixture model. We also have, corresponding to each point in $\mathcal{X}$ a timestamp (and similarly for $\mathcal{Y}$). Let t_i be the timestamp corresponding to x_i, and s_i the timestamp corresponding to y_i. We also assume there are unobserved variables $Z_1, \dots, Z_n$ that specify which mixture component the corresponding X_i is generated from. More specifically, we assume that the variable (X_n, T_n) has distribution conditional on $Z_n = m$ given by

$$\varphi_m(x, t|R, v, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{D/2}} \frac{1}{(2\pi\tau^2)^{1/2}} e^{-||x - Ry_m - v||^2/2\sigma^2} e^{-||t - s_m|^2/2\tau^2}$$

for $m = 1, \dots, M+1$, and that (X_n, T_n) has distribution conditional on $Z_n = M+1$ equal to

$$\varphi_{M+1}(x, t|R, v, \sigma^2) = 1/N.$$

This last component is introduced to account for noise, i.e. points that do not seem to be generated from any of the points in $\mathcal{Y}$. We also assume that the mixture weights $\pi_m = (1 - w)/M$ for $m = 1, \dots, M$ and $\pi_{M+1} = w$. The parameter w indicates the proportion of points that are believed to be noise, and is set prior to running the algorithm (one could let it be one of the parameters that are optimizied for, so that one does not need to guess what its' value is beforehand, but instead optimizie it along with the other parameters in the M step. However, we will not do this). Using the law of total probability, we find that the density of the whole mixture model is

$$\varphi(x, t|R, v, \sigma^2) = w\frac{1}{N} + (1 - w)\sum_{m=1}^{M} \frac{1}{M} \varphi_m(x, t|R, v, \sigma^2).$$

Let $\theta = (R, v, \sigma^2)$. The complete likelihood function is given by

$$L(\theta; X, T, Z) = \prod_{n=1}^{N} (\frac{w}{N})^{\chi_{(Z_n = M+1)}} \prod_{m=1}^{M} \Big(\frac{1 - w}{M} \varphi_m(x_n, t_n|\theta)\Big)^{\chi_{Z_n = m}}$$

which gives the complete log-likelihood function

$$\begin{aligned} \ell(\theta; X, T, Z) = \sum_{n=1}^{N} \Big(& \chi_{(Z_n = M+1)} \log \frac{w}{N} \\ & + \sum_{m=1}^{M} \chi_{Z_n = m} \Big(\log \frac{1 - w}{M} + \log \varphi_m(x_n, t_n|\theta) \Big) \Big). \end{aligned}$$

Taking the expectation of the log-likelihood function with respect to the distribution of the latent variables Z conditional on X, T gives

$$\begin{aligned} Q(\theta, \theta') &= \mathbb{E}_{Z|X,T,\theta'} \Big(\ell(\theta; X, T, Z) \Big) \\ &= \sum_{n=1}^{N} \sum_{m=1}^{M+1} \gamma_{mn} \Big(\log \frac{1}{M} + \log \varphi_m(x_n, t_n|\theta) \Big), \end{aligned}$$

where $\gamma_{mn} = P(Z_n = m|X, T, \theta')$. Writing out the definitions of the densities φ_m and ignoring terms that do not involve any of the parameters R, v, σ^2 we end up with

$$Q = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}\sum_{m=1}^{M}\gamma_{mn}||x_n - Ry_m - v||^2 - \log\sigma^2\frac{D}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}\gamma_{mn}$$

Note that if $f(v) = ||x-Ry-v||^2 = (x-Ry-v)^T(x-Ry-v)$, then $f(v+h)-f(v) = (2(Ry+v-x))^Th + ||h||^2$, which implies that the gradient of f with respect to v is given by $2(Ry+v-x)$. So if we take the gradient of Q with respect to v we obtain

$$\nabla_v Q = -\frac{1}{2\sigma^2}\sum_{mn}\gamma_{mn}2(Ry+v-x). \tag{2.2}$$

Setting this expression equal to zero, and solving for v, we get

$$v = \frac{1}{\sum_{m,n}\gamma_{mn}}(\sum_{m,n}\gamma_{mn}x_n - R\sum_{mn}\gamma_{mn}y_m).$$

If we define $\mu_X = \frac{\sum_{mn}\gamma_{mn}x_n}{\sum_{mn}\gamma_{mn}}$ and $\mu_Y = \frac{\sum_{mn}\gamma_{mn}y_m}{\sum_{mn}\gamma_{mn}}$ we have $v = \mu_X - R\mu_Y$. Substituting this v back into () gives us

$$Q = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}\sum_{m=1}^{M}\gamma_{mn}||\hat{x}_n - R\hat{y}_m||^2 - \log\sigma^2\frac{D}{2}\sum_{n=1}^{N}\sum_{m=1}^{M}\gamma_{mn},$$

where $\hat{x}_n = x_n - \mu_X$ and $\hat{y}_m = y_m - \mu_Y$. In order to find the R that maximizes Q we will use the following lemma [5]:

Lemma 1 *Let A be a $D\times D$ matrix. Let USV^T be the singular value decomposition of A, and let*

$$C = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & \det(UV^T) \end{pmatrix}.$$

Then the rotation matrix that maximizes $R \mapsto tr(A^TR)$ is given by $R = UCV^T$.

In order to use this lemma we must rewrite Q in an appropriate form. First note that $||\hat{x}_n - R\hat{y}_m||^2 = \hat{x}_n^T\hat{x}_n + \hat{y}_m^T\hat{y}_m - 2\hat{x}_n^TR\hat{y}_m$. So

$$\begin{aligned} Q = &-\frac{1}{2\sigma^2}\Big(\sum_{mn}\gamma_{mn}\hat{x}_n^T\hat{x}_n + \sum_{mn}\gamma_{mn}\hat{y}_m^T\hat{y}_m - \sum_{mn}\gamma_{mn}\hat{x}_n^TR\hat{y}_m\Big) \\ &- \log\sigma^2\frac{D}{2}\sum_{mn}\gamma_{mn}. \end{aligned} \tag{2.3}$$

Let $\hat{X}$ be the matrix whose rows are the vectors $\hat{x}_n^T$, $n = 1, \dots, N$, and similarly let $\hat{Y}$ be the matrix whose rows are the vectors $\hat{y}_m^T$, $1, \dots, M$. We have $\Gamma^T\mathbf{1}_m = (\sum_m\gamma_{m1}, \dots, \sum_m\gamma_{mn})^T$. Let $d(\Gamma^T\mathbf{1}_m)$ be the diagonal matrix with the entries in the vector $\Gamma^T\mathbf{1}_m$ on the diagonal. Then $\hat{X}^Td(\Gamma^T\mathbf{1}_m)$ is the matrix whose n:th row is given by $(\sum_m\gamma_{mn})\hat{x}_n^T$. It follows that the first term inside the paranthesis in (2.3)

is given by $\mathrm{tr}(\hat{X}^T d(\Gamma^T \mathbf{1}_m)\hat{X})$. Similarly, the second term inside the paranthesis in (2.3) is given by $\mathrm{tr}(\hat{Y}^T d(\Gamma \mathbf{1}_n)\hat{Y})$. Finally, for the third term inside the parenthesis in (2.3) first note that $R\hat{Y}^T$ is the matrix whose columns are the vectors $R\hat{y}_m$ for $m = 1, \ldots, M$. It follows that the matrix $\hat{Y}R^T\hat{X}^T$ has entries $(R\hat{y}_m^T)^T\hat{x}_n = \hat{x}_n^T R\hat{y}_m$. Then $\Gamma^T \cdot \hat{Y}R^T\hat{X}^T$ has entries $\sum_m \gamma_{mk}\hat{x}_\ell^T R\hat{y}_m$. This implies that $\mathrm{tr}(\Gamma^T \cdot \hat{Y}R^T\hat{X}^T) = \sum_{mn} \gamma_{mn}\hat{x}_n^T R\hat{y}_m$, which is the third term. So

$$\begin{aligned} Q = &-\frac{1}{2\sigma^2}(\mathrm{tr}(\hat{X}^T d(\Gamma^T \mathbf{1}_m)\hat{X}) + \mathrm{tr}(\hat{Y}^T d(\Gamma \mathbf{1}_n)\hat{Y}) - 2\mathrm{tr}(\Gamma^T \hat{Y}R^T\hat{X}^T)) \\ &- \log\sigma^2 \frac{D}{2}\sum_{mn}\gamma_{mn}. \end{aligned} \tag{2.4}$$

Maximizing Q with respect to R is equivalent to maximizing the term $\mathrm{tr}(\Gamma^T\hat{Y}R^T\hat{X}^T) = \mathrm{tr}((\hat{X}^T\Gamma^T\hat{Y})^T R)$, which is now in the form required by the lemma, with $A = \hat{X}^T\Gamma^T\hat{Y}$. So the optimal R is given by

$$R = UCV^T, \tag{2.5}$$

where USV^T is the singular value decomposition of $\hat{X}^T\Gamma^T\hat{Y}$, and C is as in Lemma 1. Finally, we optimize the parameter σ^2. The partial derivative of Q with respect to σ^2 is given by

$$\begin{aligned} \frac{\partial Q}{\partial \sigma^2} = &\frac{1}{2(\sigma^2)^2}(\mathrm{tr}(\hat{X}^T d(\Gamma^T \mathbf{1}_m)\hat{X}) + \mathrm{tr}(\hat{Y}^T d(\Gamma \mathbf{1}_n)\hat{Y}) - 2\mathrm{tr}(\Gamma^T \hat{Y}R^T\hat{X}^T)) \\ &- \frac{1}{\sigma^2}\frac{D}{2}\sum_{mn}\gamma_{mn}. \end{aligned}$$

Setting this expression equal to zero and solving for σ^2 gives us

$$\sigma^2 = \frac{1}{D\sum_{mn}\gamma_{mn}}(\mathrm{tr}(\hat{X}^T d(\Gamma^T \mathbf{1}_m)\hat{X}) + \mathrm{tr}(\hat{Y}^T d(\Gamma \mathbf{1}_n)\hat{Y}) - 2\mathrm{tr}(\Gamma^T \hat{Y}R^T\hat{X}^T)). \tag{2.6}$$

Summarizing, we initialize the parameters by setting $R = I$ (the identity matrix), $v = 0$ and $\sigma^2 = \frac{1}{DNM}\sum_{mn}||x_n - y_m||^2$ (here D is the spatial dimension of the points, N is the number of points in $\mathcal{X}$ and M the number of points in $\mathcal{Y}$). In the M step we compute the conditional membership probabilites using Bayes' law:

$$\begin{aligned} \gamma_{mn} = P(Z_n = m|X, \theta^{(r)}) &= \frac{\varphi_m(x_n, t_n|\theta^{(r)})}{\sum_j \varphi_j(x_n, t_n|\theta^{(r)}) + c'} \\ &= \frac{e^{-||x_n - R^{(r)}y_m - v^{(r)}||/2(\sigma^2)^{(r)}} \cdot e^{-|t_n - s_m|/2\tau^2}}{\sum_j e^{-||x_n - R^{(r)}y_j - v^{(r)}||/2(\sigma^2)^{(r)}} \cdot e^{-|t_n - s_j|/2\tau^2} + c}. \end{aligned} \tag{2.7}$$

and obtain the Q function (2.4). In the E step we optimize the parameters in order to maximize Q. We obtain (2.5), (2.2) and (2.6). Substituting these values into Q gives us the maximial value:

$$\begin{aligned} Q(\theta^{(r+1)}, \theta^{(r)}) = -\frac{D\sum_{mn}\gamma_{mn}}{2}\Big(&1 + \log(\mathrm{tr}(\hat{X}^T d(\Gamma^T \mathbf{1}_m)\hat{X}) \\ &+ \mathrm{tr}(\hat{Y}^T d(\Gamma \mathbf{1}_n)\hat{Y}) - 2\mathrm{tr}(S^T C))\Big), \end{aligned} \tag{2.8}$$

where in the last term we used that $R\Gamma\hat{Y}^T = USV^T$ and $R = UCV^T$.

These two steps are iterated until either the difference $|Q(\theta^{(r+1)}, \theta^{(r)}) - Q(\theta^{(r)}, \theta^{(r-1)})|$ becomes smaller than some prespecified positive number ϵ, or until the number of iterations becomes larger than prespecified number K.

2.2 Longest Common Subsequence

For the purpose of merging, a merging of two arbitrary trajectories is justifiable if and only if there exist a common subsequence between the two trajectories. Since established earlier, each camera has its own coordinate system, with different reach, the only justifyable points for comparison between trajectories are the points in the intersected polygon between camera k and camera l. Hence, let P^k and P^l be the polygon created by all the points of the same type of traffic object in camera k respective l and let the intersection between the two be defined as $P^{i \cap j} = P^i \bigcap P^j$.

The chosen algorithm is a modified version of Longest Common Subsequence (LCSS) [6]. Furthermore, let $L_{\epsilon,T,\xi}(A, B)$ denote the longest common subsequence between two arbitrary time sorted trajectories A, B in camera k respective l. Then

$$L_{\epsilon,T,\xi}(A,B) = \begin{cases} 1 + L_{\epsilon,T,\xi}(A_{n-1}, B_{m-1}), \\ \quad \text{if } ||\boldsymbol{a}_{n-1} - \boldsymbol{b}_{m-1}|| < \epsilon \ \wedge \ |a_{t_{n-1}} - b_{t_{m-1}}| < T \\ 0 \ \text{ if } \ (A = \emptyset \vee B = \emptyset) \vee (|A| < \xi \vee |B| < \xi, \ \forall \boldsymbol{a}, \boldsymbol{b} \in P^{k \cap l}) \\ \max(L_{\epsilon,T,\xi}(A_{n-1}, B), L_{\epsilon,T,\xi}(A, B_{n-1})), \ \text{ otherwise} \end{cases} \tag{2.9}$$

where, $\boldsymbol{a}_n$, $\boldsymbol{b_m}$ are the coordinates for the nth respective mth point in trajectory A and B, ϵ is the minimal Euclidean radius between each point in meters, T is the maximum eligible time difference in seconds and ξ is the minimal length for a trajectory containg points in $P^{k \cap l}$. The reason for the inclusion of ξ is that a similarity measurement is pretty nonsensical if the amount of points are too small for one or both of the trajectories.

In addition, let the similarity between trajectories A^k in camera k and trajectory B^l in camera l be $\eta(A^k, B^l)$ and is defined as

$$\eta(A^k, B^l) = \frac{L_{\epsilon,T,\xi}(A^k, B^l)}{\min(|A^k|, |B^l|)}, \ \forall \boldsymbol{a}, \boldsymbol{b} \in P^{k \cap l} \tag{2.10}$$

thus, $\eta \in [0, 1]$ and the closer η is to 1, the more 'similar' trajectory A^k is to trajectory B^l with respect to $P^{k \cap l}$ and vice versa. Notice that (2.9) and (2.10 can also be perfermed as a similarity measurements for velocity rather than position. More details will come in the method. Furthermore, there is an additional aspect to scrutinize. Which values of η justifies a merging? Let $\bar{\eta}$ be defined as the threshold for the similarity score. Then a merging of trajectory A^k and trajectory B^l is justified if $\eta(A^k, B^l) \geq \bar{\eta}$. The choice of appropriate $\bar{\eta}$ will be done through experimentation.

To compare each ith trajectory in camera k to each jth trajectory in camera l with the addition of finding all points in P^{kl} would result into a massive time complexity and unneccessary calculations. Note, however, that $L_{\epsilon,T,\xi}(A^k, B^l)$ (and by extension $\eta(A^k, B^l)$) is nonzero if and only if atleast $|a_{ti-1} - b_{tj-1}| < T$ in eq(2.9) is satisfied. Hence, a direction to improve the time complexity, it is helpful to first find when each trajectory from one camera collide with trajectories from the other one. Therefore, define $\delta_j(A_i, B_j)$ as an auxilary function for finding if any points of trajectory B_j is in the interval of trajectory A_i, defined as

$$\delta_j(A_i, B_j) = \begin{cases} 1 \text{ if } B_j \in [A_{t_1}, B_{t_n}] \\ 0 \text{ otherwise.} \end{cases}$$

2.3 Merging

When $\eta(A_i^k, B_j^l)$ is above a certain threshold of LCSS score, $\bar{\eta}$ it is sufficient to merge the two trajectories A_i^k and B_j^l. The merging consists of finding the LCSS array from Alg(1) between A_i^k and B_j^l and use backtracking to find where the points are matched. Let therefore $\Theta(A_i^k, B_j^l)$ be the merging function between the ith trajectory in camera k and jth trajectory in camera l. Consequently, let $\alpha \in A^k$ be the matched points from trajectory A^k and $\beta \in B^l$ be the corresponding matched points in trajectory B^l. Then, the coordinates becomes the point inbetween them and the merged time is the minimum of them, added with the middle time difference between the matched point. Therefore, the new trajectory is $\Theta(A_i^k, B_j^l) = \{\theta_1, \theta_2, \ldots, \theta_I\}$ where

$$\theta_i = \begin{cases} (a_x, a_y, a_t), \text{ if } a_i \in A \wedge a_i \notin \alpha \\ (b_x, b_y, b_t), \text{ if } b_i \in B \wedge b_i \notin \beta \\ \zeta_i = (\frac{a_x+b_x}{2}, \frac{a_y+b_y}{2}, \min(a_t, b_t) + \frac{|a_t-b_t|}{2}), \text{ otherwise.} \end{cases} \tag{2.11}$$

It should be clear that the points $\{\theta_1, \theta_2, \ldots, \theta_I\}$ are time-sorted, similar to A^k and B^l and ζ_i is a matched point.

In the case of a situation which involves more than two cameras; let $k_1, k_2 \ldots k_N$ denote the kth camera where $N > 2$ be the number of maximum cameras in the data set, then φ_i be the ith merging over $i \leq N-1$. Furthermore, the merging φ_i with respect to polygon $P^{(k_1,k_2,\ldots,k_{i-1})\cap k_i} = (P^{k_1} \bigcup P^{k_2} \bigcup \cdots \bigcup P^{k_{i-1}}) \bigcap P^{k_i}$ is recursive and defined as $\varphi_i = \Theta(\varphi_{i-1}, A^{k_{i+1}}), \ \forall i \leq N-1$ since

$$\begin{aligned}
&\varphi_0 = \Theta(A^{k_1}, \emptyset) = A^{k_1}, \ \forall \zeta_i \in P^{k_1} \\
&\varphi_1 = \Theta(A^{k_1}, A^{k_2}) = \Theta(\Theta(A^{k_1}, \emptyset), A^{k_2}) = \Theta(\varphi_0, A^2), \ \forall \zeta_i \in P^{k_1 \cap k_2} \\
&\varphi_2 = \Theta(A^{k_1}, A^{k_2}, A^{k_3}) = \Theta(\Theta(A^{k_1}, A^{k_2}), A^{k_3}) = \Theta(\varphi_1, A^{k_3}), \ \forall \zeta_i \in P^{(k_1,k_2)\cap k_3} \\
&\vdots \\
&\varphi_i = \Theta(\varphi_{i-1}, A^{k+1}), \forall \zeta_i \in P^{(k_1,k_2,\ldots,k_i)\cap k_{i+1}} \\
&\vdots \\
&\varphi_{N-1} = \Theta(\varphi_{N-2}, A^{k_N}), \forall \zeta_i \in P^{(k_1,k_2,\ldots k_{N-1})\cap k_N}
\end{aligned}$$

where A^{k_i} is trajectory A in camera k_i and $\boldsymbol{\zeta}$ is all paired points which satisfies the third row in (2.11). Note that the merging order is not always commutative when $N > 2$ even though the first merging step does commute. When $N = 3$ there are 6 viable merging permutations $\Theta(A^{k_1}, A^{k_2})$,$\Theta(A^{k_1}, A^{k_3})$, $\Theta(A^{k_2}, A^{k_3})$, $\Theta(A^{k_1}, A^{k_2}, A^{k_3})$, $\Theta(A^{k_1}, A^{k_3}, A^{k_2})$ and $\Theta(A^{k_2}, A^{k_3}, A^{k_1})$ which is pretty harmless computationaly. However when $N = 4$ there are a total of 30 possible permutations and it scales pretty quickly from there. There is nonetheless only a need to

check all possible permutation if an overall area $\bar{P} = \bigcap_{i=1}^{N} P^{k_i}$ exists for the chosen traffic object. When $\bar{P} = \emptyset$ then the amount of permutation can be reduced sufficiently, especially for $N > 3$. As a consequence, if the different polygons are aligned in such a way that $P^{k_1} \bigcap P^{k_2}, P^{k_2} \bigcap P^{k_3}, \ldots, P^{k_{N-1}} \bigcap P^{k_N}$ or the trajectories in each polygon are moving in such an order; then the merging orders can be reduced to a single order, either $\Theta(A^{k_1}, A^{k_2}, \ldots, A^{k_N})$ or $\Theta(A^{k_N}, A^{k_{N-1}}, \ldots, A^{k_1})$. Both of which will yield the same results.

3
Method

The choice of programming language is Python with *numpy*, *pandas* and *Polygon* from *shapely.geometry* where the last one is mainly being used in handling of the different polygons.

3.1 LCSS

Let A and B be trajectories of known traffic object, the LCSS algorithm can, irrespective of P^{kl} be described as

Algorithm 1: LCSS algorithm, $L_{\epsilon,T,\xi}$: Finding the longest common subsequence.

Input: A, B, ϵ, T, ξ
Output: LCSS array

$N \leftarrow len(A)$
$M \leftarrow len(B)$
$D \leftarrow zeros(N+1, M+1)$
if $N < \xi$ *or* $M < \xi$ **then**
    return D
else
    for $i = 1, \dots N$ **do**
        for $j = 1, \dots M$ **do**
            $x \leftarrow ||A[i-1, :-1] - B[j-1, :-1]||$
            $\Delta t \leftarrow |A[i, -1] - B[j, -1]|$
            if $x < \epsilon$ *and* $\Delta t < T$ **then**
                $D[i, j] := 1 + D[i-1, j-1]$
                else
                $D[i, j] := \max(D[i-1, j], D[i, j-1])$
    return D

where the last element $D[-1, -1]$ is the largest common subsequence. To find where the matched points occur, use backtracking.

When taking the data frame of the cameras into consideration, let C_k be the data collected through camera k, containg ID numbers, coordinates (x, y), time and of the same type of traffic object. Furthermore, let $A_i^k \in C_k$ be the ith trajectory in camera k. Thus $C_k = \{A_1^k, \dots A_n^k\}$. Note that for this section, each point $\boldsymbol{a} = (x, y)$ are the coordinates in terms of the global coordinate system. Let $H_{ij}^{kl} = \eta(A_i^k, A_j^l)$ denote

the similarity matrix and is the resulting matrix of comparing the ith trajectory in camera C_k with the jth trajectory in camera C_l, where $\eta(A_i^k, A_j^l)$ is defined in (2.10). Thus, the algorithm to find matrix H is

Algorithm 2: Similarity matrix H

Input: $C_k, C_l, \epsilon, T, \xi, P^{kl}$
Output: H

$U_2 \leftarrow C_l.ID.unique()$
$M \leftarrow len(U_2)$
$H \leftarrow list()$
for $A \in C_k$ **do**
  Find all indices $iTmp$ in A where $\boldsymbol{a} \in P^{kl}$
  $A \leftarrow A[iTmp]$
  $row \leftarrow zeros(M)$
  if $A \neq \emptyset$ **then**
    $\delta \leftarrow \{p_1, \ldots, p_m\}, \ \forall p_j \in C_l[A[0,-1], A[-1,-1]]$
    $\tilde{\boldsymbol{B}} \leftarrow \delta.ID.unique()$
    if $\tilde{\boldsymbol{B}} \neq \emptyset$ **then**
      for $B \in \tilde{\boldsymbol{B}}$ **do**
        Find all indices $iTmp$ in B where $\boldsymbol{b} \in P^{kl}$
        $B \leftarrow B[iTmp]$
        if $B \neq \emptyset$ **then**
          $L \leftarrow L_{\epsilon,T,\xi}(A, B)$
          $row[U_2 == B] \leftarrow \frac{L[-1,-1]}{\min(len(A), len(B))}$
  $H.append(row)$
return H

Since the aim is to merge a unique trajectory in C_k with another unique trajectory in C_l, that is to say, each row should be uniquely assigned to another column in H^{kl}, which results into the assignment problem. This paper uses the Hungarian algorithm [7] to find the optimal association between each traffic object, similar to [3]. Afterwards filtered out the paired trajectories which have a similarity score equal to zero. The consequences of this is an array Λ^{kl} containg the ID's from camera k and camera l and their score which will be necessary when merging.

3.2 Merging

To determine which trajectories worth merging, it facilitates to have a list over trajectories and their similarity score. Let $\tilde{A}_i^k$ be the ith trajectory in camera k and $\tilde{A}_i^l$ be the corresponding matched trajectory in camera l. Then Λ^{kl} is defined as

$$\Lambda = \begin{pmatrix} \tilde{A}_1^k & \tilde{A}_1^l & \eta(A_1^k, A_1^l) \\ \tilde{A}_2^k & \tilde{A}_2^l & \eta(A_2^k, A_2^l) \\ \vdots & \vdots & \vdots \\ \tilde{A}_n^k & \tilde{A}_n^l & \eta(A_n^k, A_n^l) \end{pmatrix}. \tag{3.1}$$

where $\eta(A_1^k, A_1^l) > \eta(A_2^k, A_2^l), \ldots, > \eta(A_n^k, A_n^l)$. To create a merging data frame, Φ_{kl}, using Λ^{kl}, define the merging algorithm

Algorithm 3: Merging trajectories from two data frames based on $\bar{\eta}$.

Input: $C_k, C_l, \Lambda^{kl}, \bar{\eta}$
Output: φ_{kl}

$i = 0$
$\Phi_{kl} = list()$
while $\Lambda_{i3}^{kl} > \bar{\eta}$ *and* $i < |\Lambda^{kl}|$ **do**
    $\Phi_{kl}.append(\Gamma(\Lambda_{i1}^{kl}, \Lambda_{i2}^{kl}))$ `// Merge the two trajectories using 2.11 and then append them into a merged data frame.`
    $i = i + 1$
return Φ_{kl}

where each point in C_k and C_l consist of coordinates and time. According to Alg(3), a merge between trajectory A_i^k and A_i^l will thus only be executed if their similarity score is larger than the similarity threshold $\bar{\eta}$, i.e $\eta(A_i^k, A_i^l) \geq \bar{\eta}$. Additionally, since every traffic scene is unique and the amount of permutation is dependent on how the different polygons intertwines with one another, define Ω as a list of merging order where the first two elements in each row in Ω indicates which two cameras will be merged first and then be compared and merged with the third element and so forth. The process will then be repeated in the next row in Ω. In conclusion, the merging process between N cameras, given that $\boldsymbol{C} = \{C_1, C_2, \ldots C_N\}$, is therefore

Algorithm 4: Merging between all cameras.

Input: $\boldsymbol{C}, T, \epsilon, \xi, \bar{\eta}, \Omega$
Output: $\Phi = \{\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, \ldots \boldsymbol{\varphi}_{N-1}\}$

$\Phi = list()$
for *order in* Ω **do**
    $H^{ij} \leftarrow SimilarityMatrix(C_i, C_j, T, \epsilon, P^{C_i \bigcap C_j}, \xi)$
    Do hungarian on H^{ij} and find Λ^{ij}
    $\varphi_{ij} \leftarrow CreateMergedDataFrame(C_i, C_j, \Lambda^{ij}, \bar{\eta})$
    $\Phi.append(\varphi)$
Remove duplicates φ_i's if $|\varphi_{ij}| \in |\varphi_{ijk}|$.
return Φ

3.3 TimeCPD

In this section we describe the implementation of the CPD algorithm, and also the pre-computations done in order to reduce the amount of noise in the data.

All the points in the dataset $\mathcal{X}$ recorded by the first camera do not correspond to some point in the data set $\mathcal{Y}$ recorded by the second camera. This is because the portions of the scene recorded by the different cameras are not the same, but only partially overlap. This means that the points in $\mathcal{X}$ that lie outside the overlapping region do not match to any points in $\mathcal{Y}$, and vice versa. So only the points that lie in the overlapping region contribute to finding the correct rigid transformation

between the coordinate systems of the different cameras; all points that lie outside this region can be consider as "noise". The (modified) CPD algorithm is robust to a certain amount of noise, but since the nonoverlapping regions of the different cameras are (usually) larger than the overlapping region, the amount of "noise" can be quite large.

Unfortunately we do not know beforehand which points belong to the overlapping region. So before any of the algorithm below is performed, with the purpose to reduce noise, a modified version of Alg(1), Alg(2) based on time and velocity rather than time and position is performed. Third column in (3.1) will instead consist of a similarity score based on velocity. Furthermore instead of merging data frames in Alg(3), backtracking is done in order to find which points are similar in terms of speed. Thus the output of this modified version of Alg(3) will give an output of $\boldsymbol{\zeta}_{kl}^{speed}$ which contains all the matched points between trajectories which had a higher similarity score than $\bar{\eta}^{speed}$ in camera k and camera l.

Below is pseudocode describing our implementation of the CPD algorithm. The main function TimeCPD takes as input the data sets X and Y (whose rows are of the form (x, y, z, t), where (x, y, z) are the spatial coordinates of a point, and t is the time when the point was recorded by the camera), maxIter (maximum number of iterations), tol (parameter controlling convergence...), tau2 (variance of the distribution functions for the timestamps), and w (parameter describing the amount of noise believed to be in the data). We set N equal to the number of rows in X, M equal to the number of rows in Y and $D = 3$, the spatial dimension. First, the parameters that will be optimized for are initialized. We set $R = I$, $t = 0$ and $\sigma^2 = \frac{1}{DNM}\sum_{mn}||x_n - y_m||^2$. Also, the variable currIter (the number of iterations performed) is initialized to 0, the variable err (the current absolute value of the difference between Q for the current iteration and the last iteration) is initialized to ∞, and the variable Qprev (the value of Q in the last iteration) is initialized to ∞. Next there is a while loop, inside which the two steps (E step and M step) of the EM algorithm are carried out, either until convergence occurs, or the number of iterations exceeds maxIter. Inside the while loop, first the points in Y are transformed using the current rotation R and translation v. The transformed points TY, along with X, w, σ^2, τ^2, are input into a function that performs the E step, and returns the matrix Γ containing the conditional membership probabilites. This matrix, along with the data sets X and Y are then sent to a function performing the M step. This function returns the new values of the parameters R, v and σ^2, along with the optimal value of the objevtive function Q. Finally, the error is computed, Qprev is set equal to the new optimal value of Q, and currIter is incremented, and we go to the next iteration. After we exit the while loop (either because of convergence or because we have exceeded the maximal number of iterations allowed) we transform the points in Y using the final values of the parameters and return the rotation R,

the translation v, along with some other things.

Algorithm 5: TimeCPD

Input: X,Y, maxIter, tol, w, τ^2
Output: R, v

Initialize R to the identity matrix, v to the zero row vector, and $\sigma^2 = \frac{1}{DNM}\sum_m n||x_n - y_m||^2$
Initialize currIter to zero, err to ∞, and Qprev to ∞
while *currIter* $\leq$ *maxIter and err* $\geq$ *tol* **do**
    TY = TransformPoints(Y, R, v)
    Γ = Estep(X, TY, w, σ^2)
    R, v, σ^2, Q = Mstep(X, Y, Γ)
    err = |Q-Qprev|
    Qprev = Q
    currIter = currIter + 1
TY = TransformPoints(Y,R,v)

Algorithm 6: Estep

Input: X, TY, w, σ^2, τ^2
Output: Γ

Compute Γ using Bayes' law as in (2.7)

Algorithm 7: Mstep

Input: X, Y, Γ
Output: R, v, σ^2, Q, w

Compute v, σ^2 and R as in (2.2), (2.6) and (2.5)
Compute optimal Q as in (2.8)

Algorithm 8: TransformPoints

Input: Y, R, v
Output: TY

$TY = YR^T + v$

4
Results

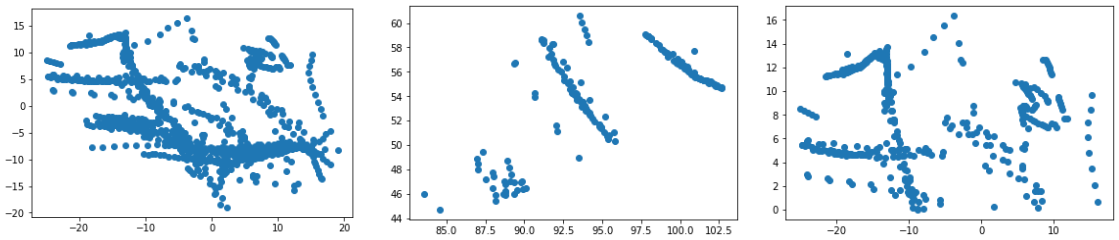
Each data poin includes their ID number, x and y coordinates, time, speed, type (pedestrian, cyclist, car or heavy vehicle) and estimation. The data is not entirely raw, meaning it has been through some kind of filtering to a different coordinate system, hence the category estimation. Estimation is binary and determines if the data point is dubious (1) or sufficient (0). If each of the last data points in an arbitrary trajectory has the estimation value 1, then it is most likely not a sufficient trajectory. However the amount of points which require to be 1 is not given. This thesis uses the minimal length trajectory ξ as the same measure of amount of points in the end of a trajectory is equal to 1 before discarding it. Hence, before any calculations are made, a trajectory is temporary discarded from the data if the sum the amount of last points (ξ) whose estimation sum are also equal to ξ.

4.1 Simulated data

In the simulated data, the proposed method shall put to test before performing data from real traffic.

4.1.1 Camera alignment

We take a subset of the data from Skövde camera 111 data set, and use the points along with the timestamps in this dataset as the point set X. Then we take all those points in X whose x coordinate lie to the left of some line $x = x_0$ andapply a random rotation matrix R and a random translation t to these points, letting this point set be Y. Finally, from the original point set X we remove all points whose y coordinate lie above a line $y = y_0$. See figure 4.1 for an example of how this might look.



(a) Original subset of Skövde camera 111 data

(b) Dataset Y

(c) Dataset X

Figure 4.1: Datasets used for testing (subplots (b) and (c)) obtained from subset of Skövde camera 111 data (subplot (a))

Using the obtained datasets X and Y as input to timeCPD we in practically all cases attempted got results very close to the ground truth (i.e. the random rotation R and the random translation t mentioned above used to obtain Y). For example, in figure 4.2 we can see the result when timeCPD is applied to the datasets in figure 4.1. In this case the rotation R is roughly equal to

$$\begin{pmatrix} 0.54 & 0.84 \\ -0.84 & 0.54 \end{pmatrix}$$

and the translation is roughly equal to $t = (12.76, 8.21)^T$. The rotation output from timeCPD is roughly equal to

$$\begin{pmatrix} 0.53 & 0.85 \\ -0.85 & 0.53 \end{pmatrix}$$

and the translation output from timeCPD is roughly equal to $(12.4, 8.3)^T$, which in this case is very close to the ground truth.

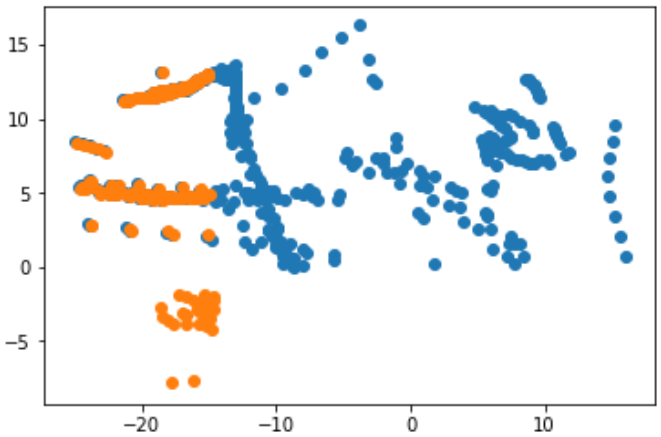


Figure 4.2: Datasets used for testing ((b) and (c)) obtained from subset of Skövde camera 111 data ((a))

4.1.2 Matching Trajectories

Let the simple function $y(x) = x^2,\ x \in [0, 6]$ be split into two. Let the first part be A where $x \in [0, 3.6]$ and the second part called B, where $x \in [3, 6]$. When performing Alg(1) with only the spatial coordinates to consider (i.e all time stamps are the same) it produced

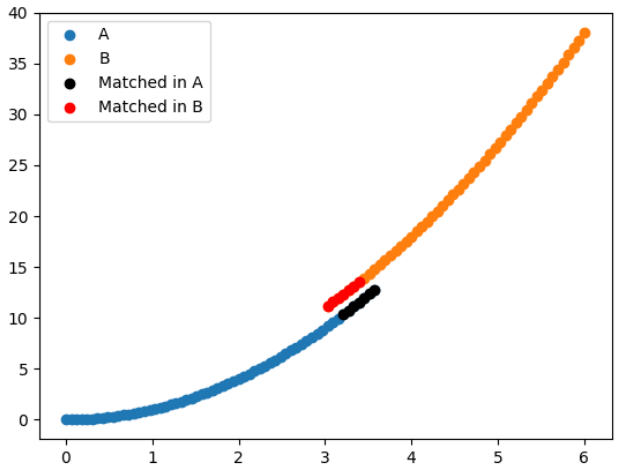


Figure 4.3: Alg(1) with only respect to coordinates. Blue: trajectory A, orange: trajectory B, black: matched points from A to B, Red: matched points from B to A.

where ϵ was chosen to 1. When taking time into consideration, T was chosen for 1 second. Furthermore, four of the matched points (black and red) was chosen randomly to have a time difference of less than one second, whereas the rest had more than one second. The result became

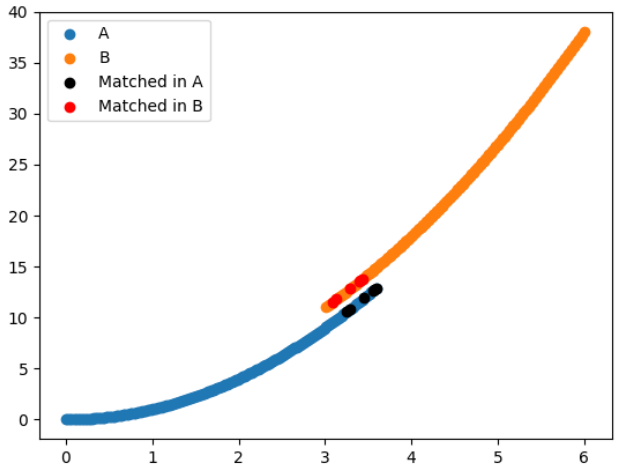


Figure 4.4: Alg(1) with respect to time and coordinates. Blue: trajectory A, orange: trajectory B, black: matched points from A to B, Red: matched points from B to A.

where Alg(1) found the three that should be matched. The next test is to see if LCSS is capable of finding the matched points with respect to a polygon

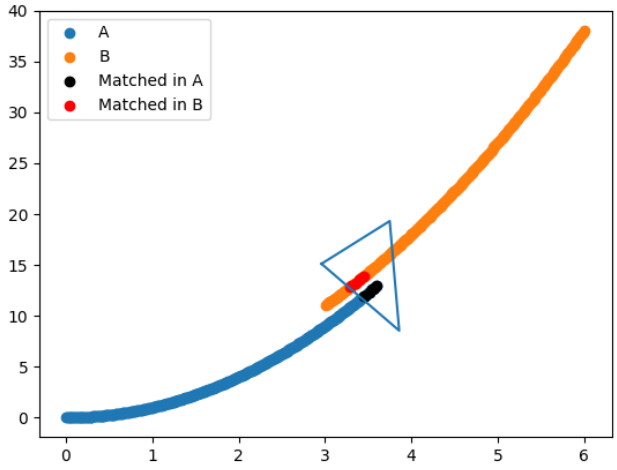


Figure 4.5: Alg(2) with respect to time, coordinates and polygon. Blue: trajectory A, orange: trajectory B, black: matched points from A to B, Red: matched points from B to A.

where the blue triangle is a polygon. The last test is for LCSS to perform with respect to an overlap. Alg(2) with respect to overlap became

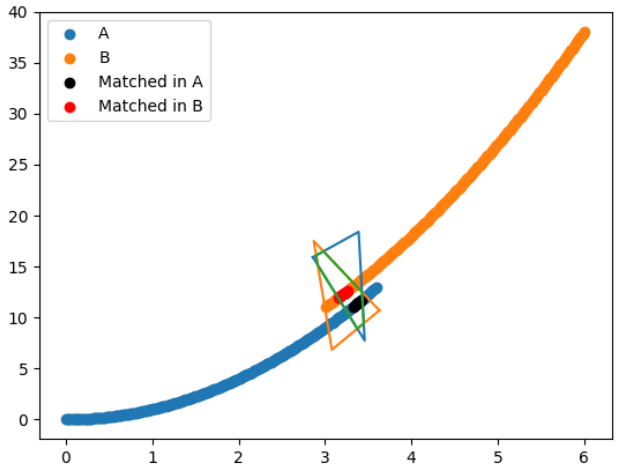
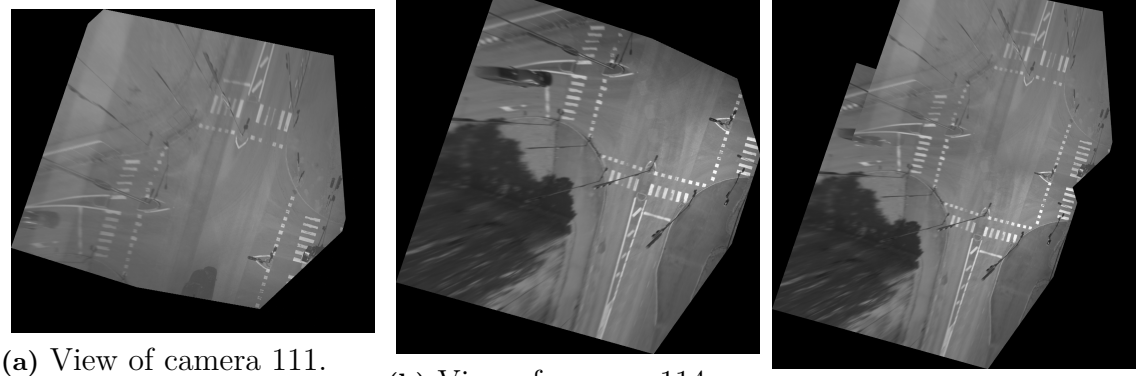


Figure 4.6: Alg(2) with respect to time, coordinates and overlap. Blue: trajectory A, orange: trajectory B, black: matched points from A to B, Red: matched points from B to A.

where the orange and blue triangle are two polygons including their overlap (green) to simulate two cameras and their overlap. Figure(4.6) shows that LCSS is capable of find matching based on space, time, and overlap in a simulated area.

4.2 Skövde Data

The first data is an intersection in Skövde, Sweden and which contains two cameras



(a) View of camera 111.

(b) View of camera 114.

(c) View of camera 111 and 114 put together.

Figure 4.7: Camera view of camera 111 and camera 114 in Skövde.

where 4.7.a and 4.7.b are the polygons of camera 111 and 114 respectively for the whole crossing 4.7.c. Camera 111 and camera 114 consist of 261584 respectively

160667 data points. The total number of trajectories after first filtering based on estimated for the intersection in Skövde was

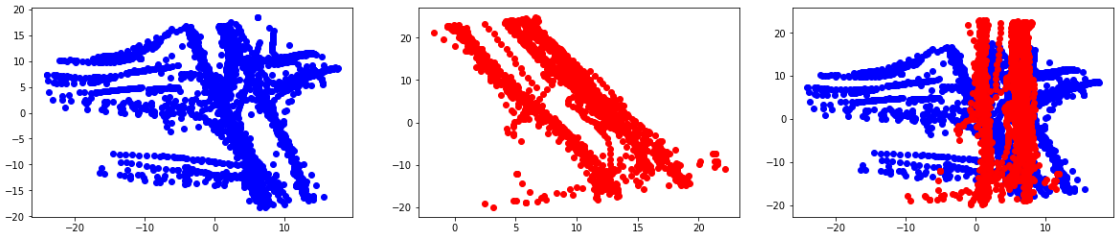
Table 4.1: Number of trajectories for each traffic object. $(*)$: number of trajectories which has atleast ξ points in $P^{111,114}$

	Camera 111 $(*)$	Camera 114 $(*)$	Total traffic object $(*)$
Pedestrians	132 (38)	136 (4)	268 (42)
Cyclists	896 (293)	475 (183)	1371 (476)
Cars	11075 (6189)	8423 (1716)	19498 (7905)
Heavy vehicles	2013 (1324)	1166 (467)	3019 (1791)
Total	14116 (7844)	10200 (2370)	24316 (10214)

where $(*)$ determines the amount of trajectories which has atleast ξ number of points in $P^{111,114}$. Since the camera 114 has less objects in $P^{111,114}$, the $(*)$ in camera 114 also becomes the maximum amount of possible merging of the whole data set.

4.2.1 Camera alignment

Let us first show an example were no precomputations were done. We let X be the cars in camera 111 whose timestamps lie between 18:00 and 18:05, and Y be the cars in camera 114 whose timestamps lie in the same time interval. The points X can be seen in Figure 4.8 (a), and the points Y can be seen in Figure 4.8 (b). After running timeCPD with these points and transforming the points in Y with the resulting rigid transformation, we get Figure 4.8 (c). Nothing in this picture is matched up, which indicates that the resulting rigid transformation is incorrect. So just picking out a subset of the given data and using it as input to timeCPD does not usually result in a rigid transformation that seems "correct".



(a) Points corresponding to cars in camera 111 whose timestamps lie between 18:00 and 18:05

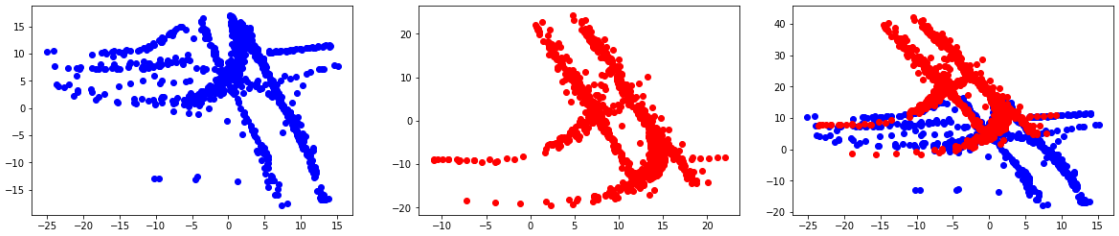
(b) Points corresponding to cars in camera 114 whose timestamps lie between 18:00 and 18:05

(c) The same points where the red points have been transformed using the rigid transformation obtained from CPD

Figure 4.8: Data without any filtering, 18:00-18:05

So in order to get better results we did some precomputations. First we divided the data set into 4 parts, corresponding to the time intervals 00:00-06:00, 06:00-12:00, 12:00-18:00 and 18:00-00:00. For each of these subdatasets we matched trajectories

using LCSS, but with velocities (which are coordinate system independent) instead of the points themselves. The minimal velocity between each point ϵ^{speed} and the maximum eligible time difference T was chosen to be 2 m/s respective 1 s was chosen to find similarities between cars in terms of speed and time. According to Table 4.1, the total data consists a total of 19498 trajectories of cars. The aim is to find as many paired trajectories with minimal of ambiguity, hence $\bar{\eta}^{speed} = 0.8$ was chosen, i.e they are much alike (in terms of speed and time). The first of these datasets contained few points, and so we decided not to include it. The results obtained for the other datasets are presented below.

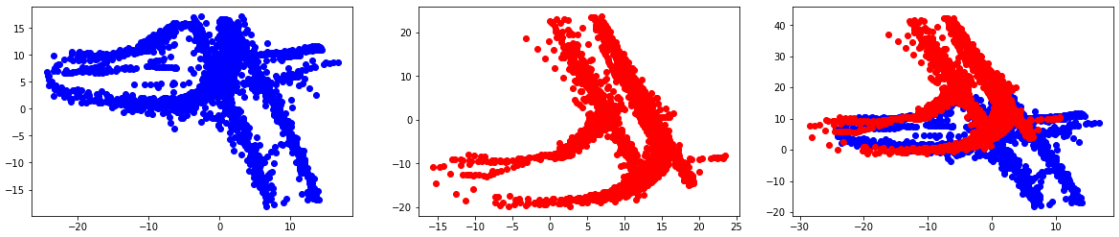


(a) Points corresponding to cars in camera 111 whose timestamps lie between 06:00 and 12:00, after filtering by pre-matching using LCSS with velocities and timestamps

(b) Points corresponding to cars in camera 114 whose timestamps lie between 06:00 and 12:00, after filtering by pre-matching using LCSS with velocities and timestamps

(c) The same points where the red points have been transformed using the rigid transformation obtained from CPD

Figure 4.9: First dataset, 06:00-12:00

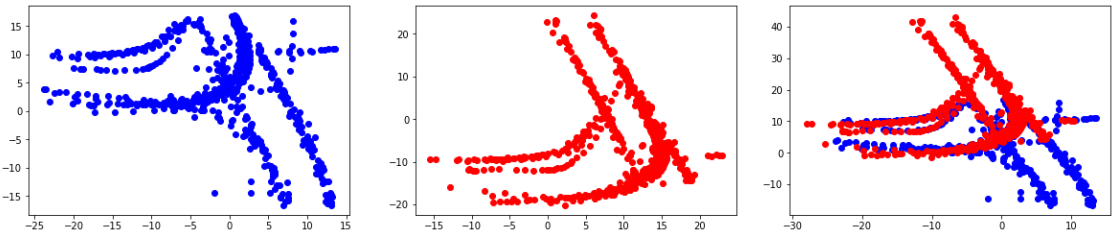


(a) Points corresponding to cars in camera 111 whose timestamps lie between 12:00 and 18:00, after filtering by pre-matching using LCSS with velocities and timestamps

(b) Points corresponding to cars in camera 114 whose timestamps lie between 12:00 and 18:00, after filtering by pre-matching using LCSS with velocities and timestamps

(c) The same points where the red points have been transformed using the rigid transformation obtained from CPD

Figure 4.10: Second dataset, 12:00-18:00



(a) Points corresponding to cars in camera 111 whose timestamps lie between 18:00 and 00:00, after filtering by pre-matching using LCSS with velocities and timestamps

(b) Points corresponding to cars in camera 114 whose timestamps lie between 18:00 and 00:00, after filtering by pre-matching using LCSS with velocities and timestamps

(c) The same points where the red points have been transformed using the rigid transformation obtained from CPD

Figure 4.11: First dataset, 18:00-00:00

For all three datasets we got R roughly equal to the identity matrix. The translations we got for the three different datasets were roughly the same: for the first one we got $v = (-13.34, 17.56)^T$, for the second one we got $v = (-12.84, 18.55)^T$, and for the last one we got $v = (-12.54, 18.73)^T$. These differences can probably be accounted for by the fact that the three datasets are different, and all contain some amount of noise. The fact that running timeCPD on all three of these datasets give roughly the same values of R and t makes us believe that these lie close to the "real" values of these parameters; if the algorithm gave the wrong answer, it would be weird if it gave the same wrong answer for three different datasets. As a semi verification of these results we plot the points used in Figure 4.9 (a) and (b) transformed into the common coordinate system that had been previously obtained by hand We see that the plot of these transformed points (Figure 4.12) looks essentially the same as the points in Figure 4.9 (c) after a reflection and a rotation.

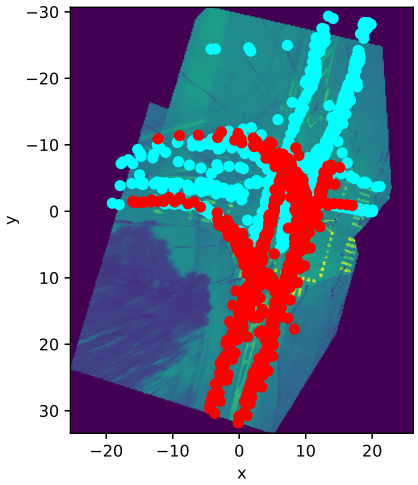


Figure 4.12: Red: Trajectories from camera 114. Cyan: Trajectories from camera 111.

4.2.2 Matching Trajectories

Based on Table(4.3), regular cars will be one of the main focus regarding overall Skövde data.

4.2.2.1 Cars

The polygons created based on all the coordinates in both cameraas became

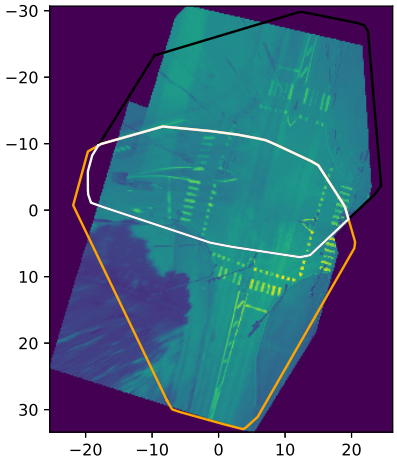
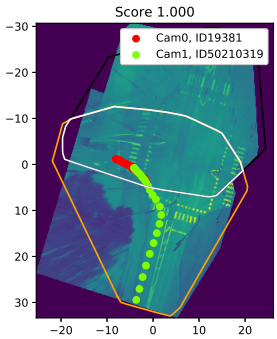
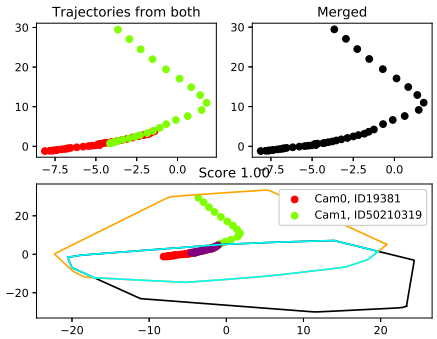


Figure 4.13: Black is the polygon created through all cars in camera 111. Orange is the polygon created through all cars in camera 114. White is the intersection of camera 111 and 114.

where the black polygon is camera 111, orange is camera 114 and white is their intersection. There were scores with maximum similiarty score such as

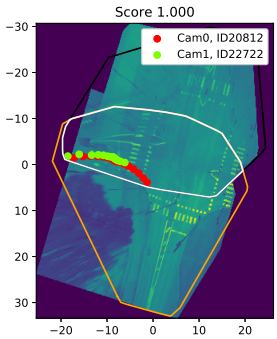


(a) Overview with polygon of 19381 and 50210319.

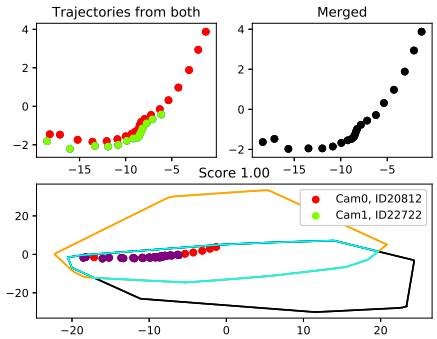


(b) How the merging between 19381 and 50210319 would look like.

Figure 4.14: Comparison between camera 111 and 114; upper left: 19381 and 50210319 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity Score: 1.0



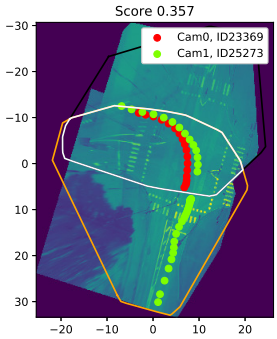
(a) Overview with polygon of 20812 and 22722.



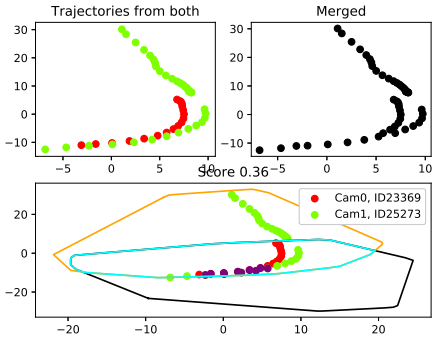
(b) How the merging between 20812 and 22722 would look like.

Figure 4.15: Comparison between camera 111 and 114; upper left: 20812 and 22722 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 1.0

where in these cases, the temporary merging is also a sufficient permanent merging of the two trajectories. The chosen threshold was $\bar{\eta} = 0.35$. Car trajectories with a score slightly above $\bar{\eta} = 0.35$ was



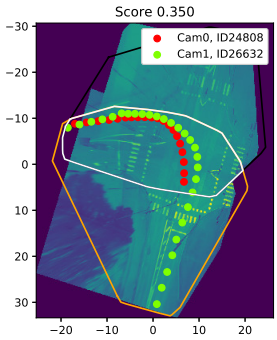
(a) Overview with polygon of 23369 and 25273.



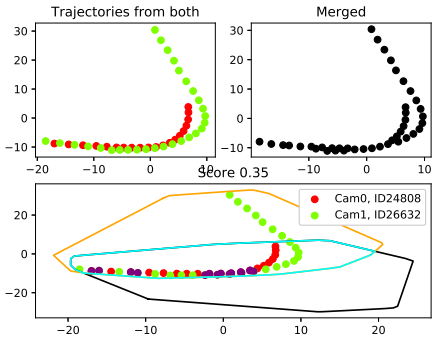
(b) How the merging between 23369 and 25273 would look like.

Figure 4.16: Comparison between camera 111 and 114; upper left: 23369 and 25273 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 0.357

is still sufficient to merge. The same argument can still be made for trajectories with equal values $\bar{\eta} = 0.35$ such as

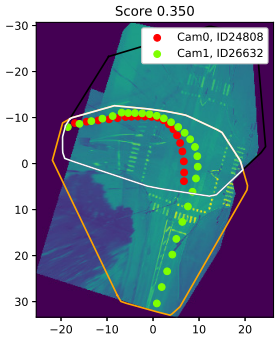


(a) Overview with polygon of 24808 and 26632.

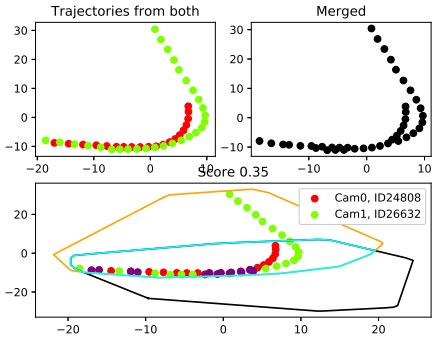


(b) How the merging between 24808 and 26632 would look like.

Figure 4.17: Comparison between camera 111 and 114; upper left: 24808 and 26632 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 0.35



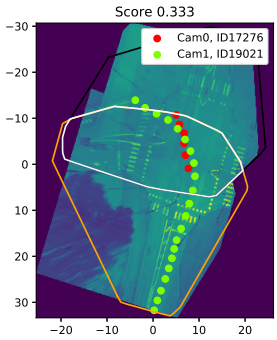
(a) Overview with polygon of 24808 and 26632.



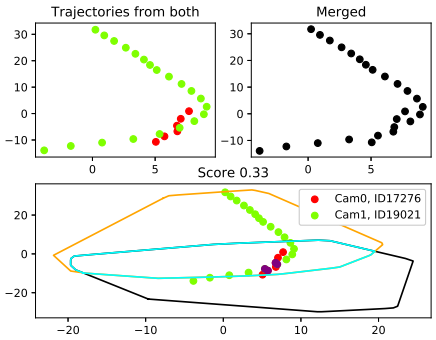
(b) How the merging between 24808 and 26632 would look like.

Figure 4.18: Comparison between camera 111 and 114; upper left: 24808 and 26632 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 0.35.

Right below $\bar{\eta} = 0.35$ the frequncey of less coherent merged trajectories are increasing such as



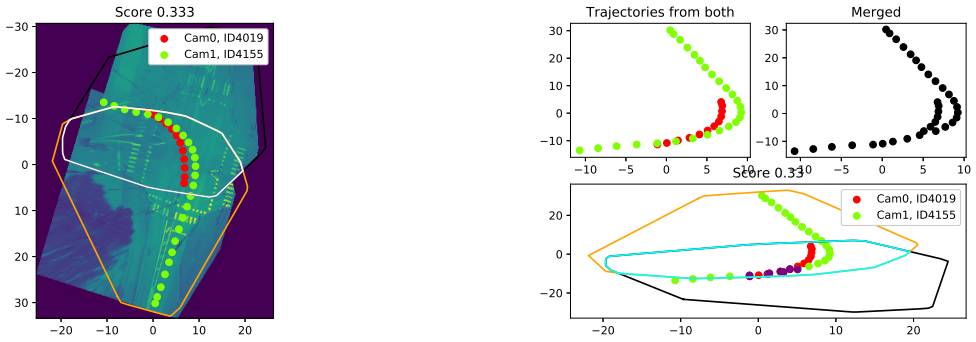
(a) Overview with polygon of 17276 and 19021.



(b) How the merging between 17276 and 19021 would look like.

Figure 4.19: Comparison between camera 111 and 114; upper left: 17276 and 19021 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 0.33.

However there also exists some justifiable ones below $\bar{\eta} = 0.35$ such as



(a) Overview with polygon of 4019 and 4155.

(b) How the merging between 4019 and 4155 would look like.

Figure 4.20: Comparison between camera 111 and 114; upper left: 4019 and 4155 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 0.333

Overall, $\bar{\eta} = 0.35$ was chosen as the threshold to add more certainty in the merging process. Note that for a pair of trajectory to have a similarity score around 0.35, the temporary merging is not sufficient.

4.2.2.2 Heavy Vehicles

The polygons for heavy vehicles

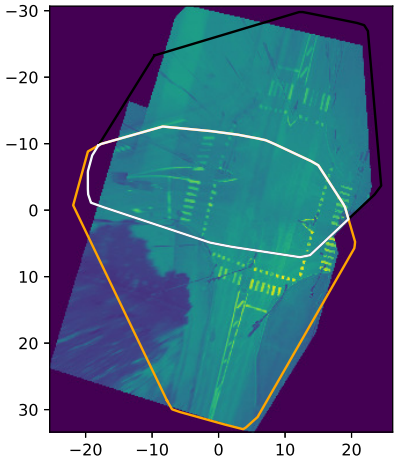
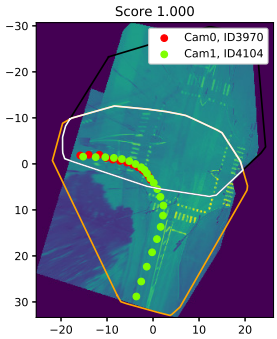
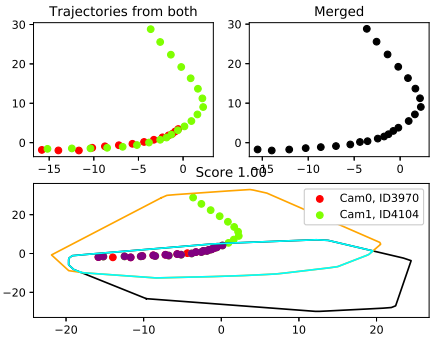


Figure 4.21: Black is the polygon created through all pedestrians in camera 111. Orange is the polygon created through all pedestrians in camera 114. White is the intersection of camera 111 and 114.

which is similar to 4.13 as it should be since both vehicles cover the same ground in the intersection as shown below.

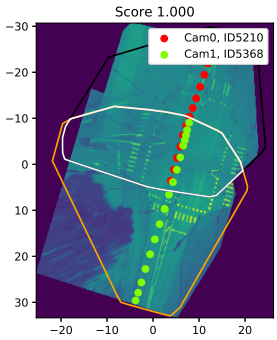


(a) Overview with polygon of 3979 and 4104.

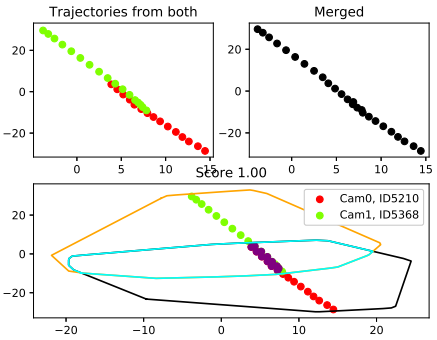


(b) How the merging between 3979 and 4104 would look like.

Figure 4.22: Comparison between camera 111 and 114; upper left: 3979 and 4104 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 1



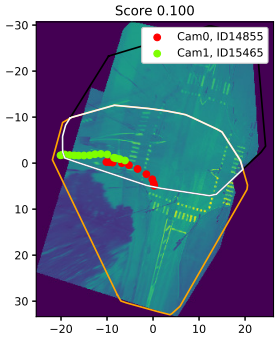
(a) Overview with polygon of 5210 and 5368.



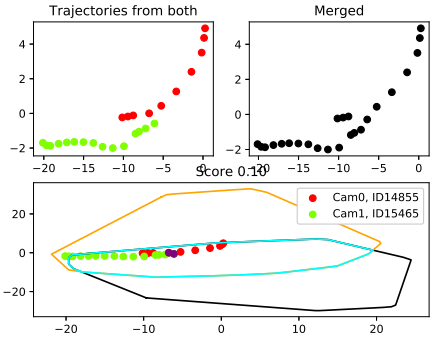
(b) How the merging between 5210 and 5368 would look like.

Figure 4.23: Comparison between camera 111 and 114; upper left: 5210 and 5368 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 1

Examples of paired trajectories with low similarity score are

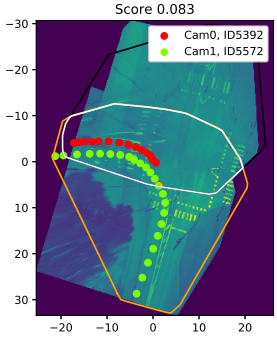


(a) Overview with polygon of 14855 and 15465.

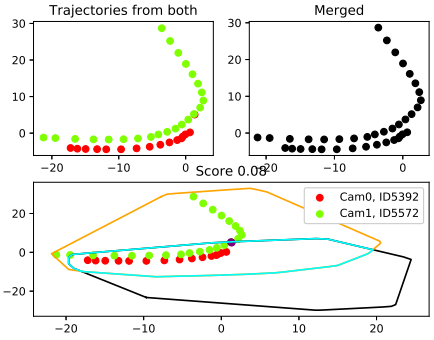


(b) How the merging between 14855 and 15465 would look like.

Figure 4.24: Comparison between camera 111 and 114; upper left: 14855 and 15465 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 0.1



(a) Overview with polygon of 5392 and 5572.



(b) How the merging between 5392 and 5572 would look like.

Figure 4.25: Comparison between camera 111 and 114; upper left: 5392 and 5572 as seperate trajectories, upper right: as merged trajectories, lower: how both trajectories interact with respect to polygon. Similarity score: 0.08

Even though trajectories follows similar pattern by distance metric, most of their time stamp do not match.

4.2.2.3 Cyclists

Created polygons for cyclists with the inclusion of all data points was

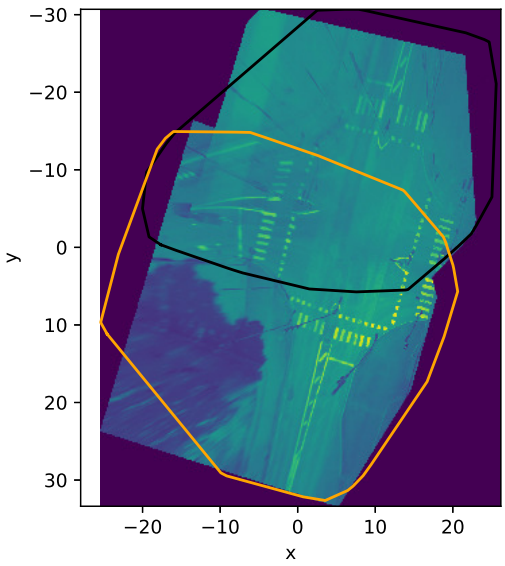
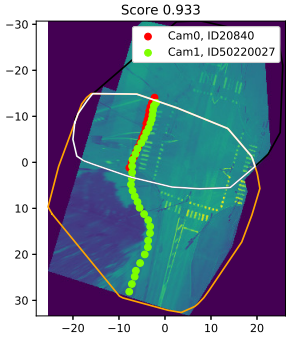
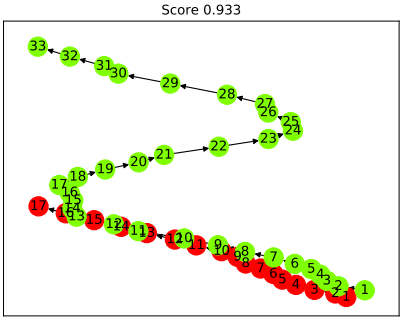


Figure 4.26: Black is the polygon created through all cyclists in camera 111. Orange is the polygon created through all cyclists in camera 114.

where the black points are the cyclists in camera 111 and orange points are cyclists in camera 114. The matched trajectories using the LCSS algorithm do also have the same direction



(a) Overview of the intersection with x and y - axis involving the trajectories interacting with camera 111 (black) and camera 114 (orange).



(b) The direction of both trajectories are the same.

Figure 4.27: Overview and a directed graph over trajectories 20840 (red) and 50220027 (green).

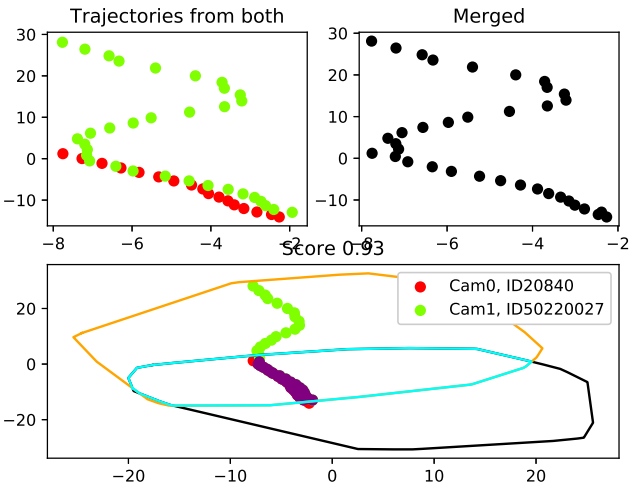


Figure 4.28: Upper left: 20840 (red) and 50220027 (green) as seperate trajectories, upper right: as merged trajectories, bottom: how both trajectories interact with respect to polygon of camera 111 (black) and camera 114 (orange). Similarity score: 0.93

as shown in figure 4.27(b) Alg 2 does take into account the direction of a pairing (otherwise it would not be a match).

4.2.2.4 Pedestrians

There was no matches were found for pedestrians. The maximum amount of merging options was 4, according to 4.3 and it shows through the intersected polygon $P^{111,114}$ for pedestrians.

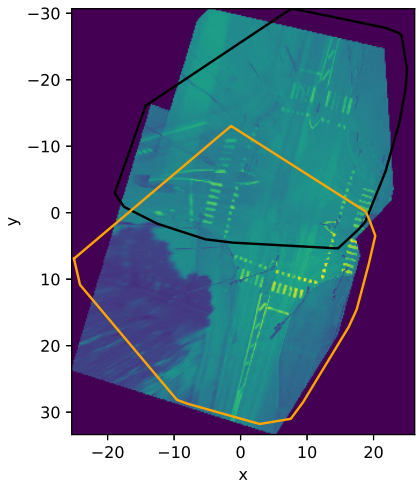


Figure 4.29: Black is the polygon created through all pedestrians in camera 111. Orange is the polygon created through all pedestrians in camera 114.

4.2.2.5 Overall

The total number of possible merging which has a similarity score above $\bar{\eta} = 0.35$ was

Table 4.2: Number of merged traffic objects with a similarity score larger or equal to than 0.35.

(ϵ, T)	Pedestrians	Cyclists	Cars	Heavy vehicles	Total
$(1, 1)$	0	12	209	72	293
$(1, 2)$	0	12	231	84	327
$(2, 1)$	0	15	922	161	1098
$(2, 2)$	0	15	945	172	1132

The Table 4.2 shows that the number of merging is more dependent on the minimal distance ϵ than time T.

4.3 Mellingen Data

The second data is a crossing in Mellingen, Switzerland and contains three cameras

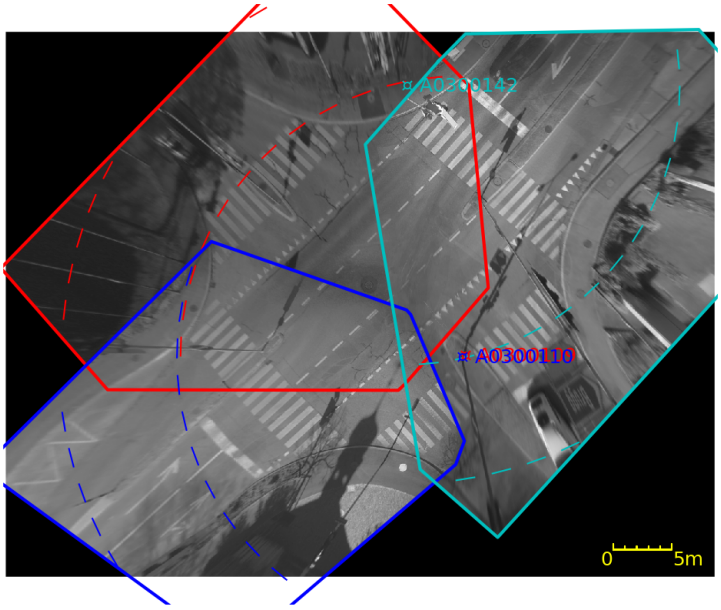


Figure 4.30: Red is the view of camera 109, blue is the view of camera 110 and cyan is the view of camera 142.

where the latter one invloves have higher number of obstacles around the intersection, which eventuates to higher density of corrupt data. Fig(4.30) is a representa-

tion of the real area the cameras cover from camera 109 (red), camera 110 (blue) and camera 142 (cyan).

Table 4.3: Number of trajectories for each traffic object for Mellingen.

	Camera 109	Camera 110	Camera 142	Total traffic object
Pedestrians	530	511	527	1568
Cyclists	409	357	497	1263
Cars	13955	12889	17133	43977
Heavy vehicles	203	264	671	1138
Total	15097	14021	18808	47926

4.3.1 Camera Alignment

As we can see in Figure 4.30, the overlapping region between cameras 110 and 142 is quite small and narrow. This lead to difficulties since most of the data will lie outside the overlapping region, resulting in large amounts of noise. An idea for finding the transformation between camera 110 and 142 is to find the transformations from camera 142 and 109, and the transformation from camera 109 to camera 110, and then compose these to get the transformation from camera 142 to camera 110. Trying to match using bycyclists or pedestrians did not work out either. This might be because the pedestrians and bycyclists would in only a small part of the overlapping regions, i.e. the crossings. For example, between camera 142 and camera 109, the crossing takes up a fairly small part of the overlap. This of course results in there being few data points corresponding to pedestrians and bycyclists in the overlapping regions. Using new data for camera 109 we ran the timeCPD algorithm again on points from camera 109 and camera 142, after a similar precomputation as before, and got the result shown in Figure 4.32 (after transforming the points into a global coordinate system, i.e. first the points in camera 142 were transformed using the rigid transformation obtained from timeCPD into the coordinate system of camera 109, and then these points along with the points from camera 109 were transformed into the global coordinate system). The rotation was roughly equal to the identity matrix and thr translation $v = (1.88, 1.09)^T$. Comparing the result with Figure 4.31, which is the points used in timeCPD transformed into the global coordinate system using the "real" transformation. The figures are quite similar, indicating that the rigid transformation found using timeCPD is close to the correct one. However, the transformed points (cyan points in Figure 4.31) are somewhat shifted downwards compared to the corresponding points in Figure 4.31. They also look as if they have been rotated slightly too much clockwise.

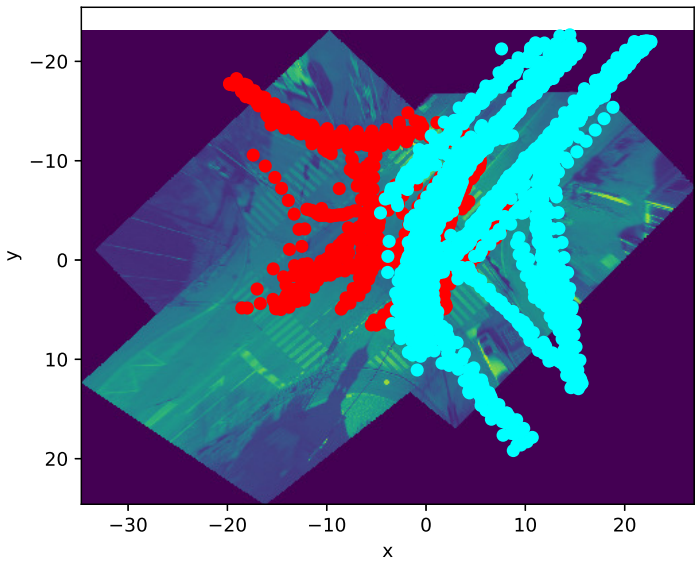


Figure 4.31: Points from camera 109 (red points), and from camera 142 (cyan points) plotted in a global coordinate system

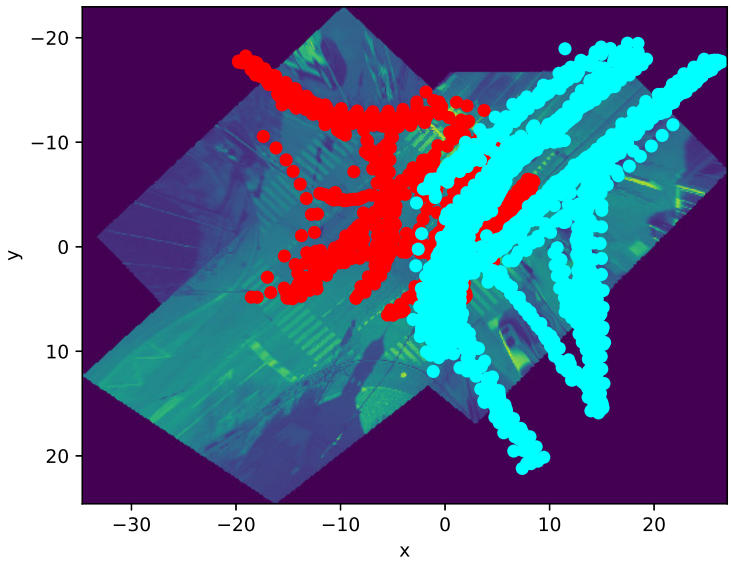


Figure 4.32: Points from camera 109 (red points), and from camera 142 (cyan points) plotted in a global coordinate system, where the points from 142 were first transformed into the coordinate system of camera 109 using the rigid transformation obtained from timeCPD

Next we also tried to find a rigid transformation between cameras 109 and 110 using the new 109 data, after once again precomputing as before. The result is show in Figure 4.34, and the "correct" plot is shown in Figure 4.33. The figures indicate

the result is very far from being correct. Looking at figure 4.33 we see that the proportion of points that overlap is quite small, meaning that most of the data can be considered "noise" when used as input to timeCPD.

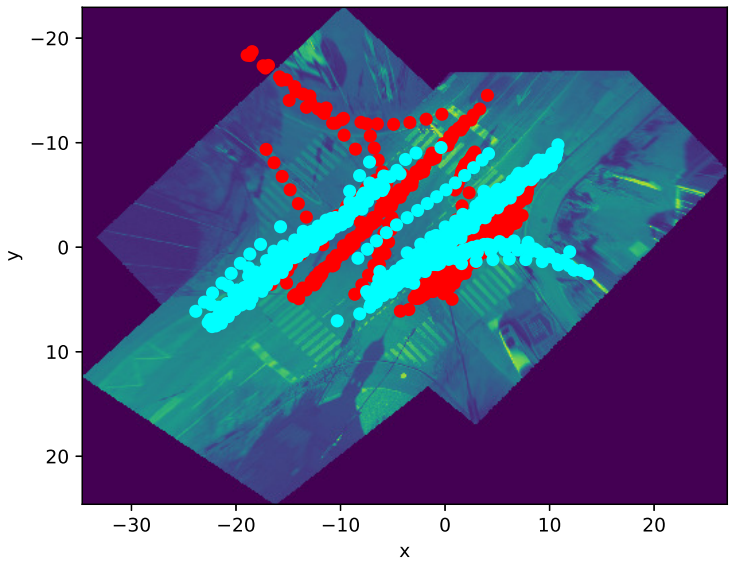


Figure 4.33: Points from camera 109 (red points), and from camera 110 (cyan points) plotted in a global coordinate system

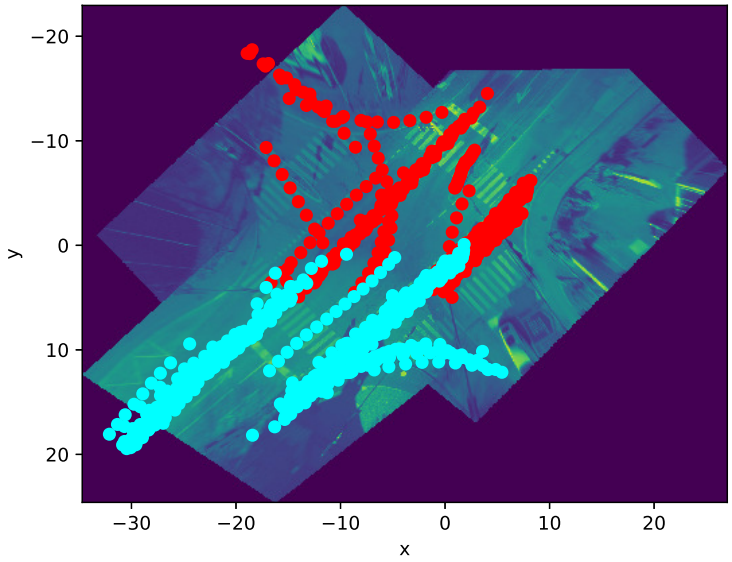


Figure 4.34: Points from camera 109 (red points), and from camera 110 (cyan points) plotted in a global coordinate system, where the points from 110 were first transformed into the coordinate system of camera 109 using the rigid transformation obtained from timeCPD

4.3.2 Matching Trajectories

4.3.2.1 Cyclists

The resulting polygons was

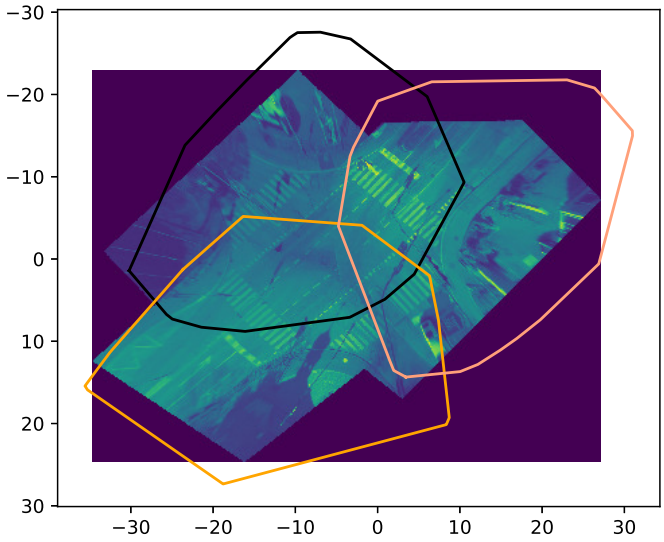


Figure 4.35: The polygons created by only using the data of cyclists. Black is camera 109, orange is camera 110 and pink is camera 142.

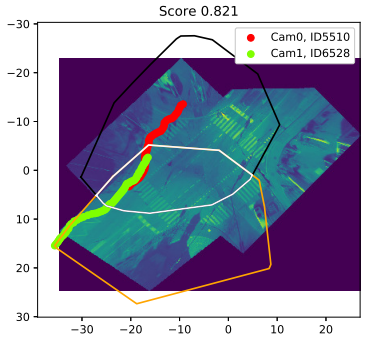
where black is for camera 109, orange for camera 110 and pink for camera 142. The number of cyclists in each camera vision with atleast ξ points in each polygon intersection can be presented in the form of a cayley table

Table 4.4: Cayley table of number of cyclists with atleast ξ points in each polygon intersection.

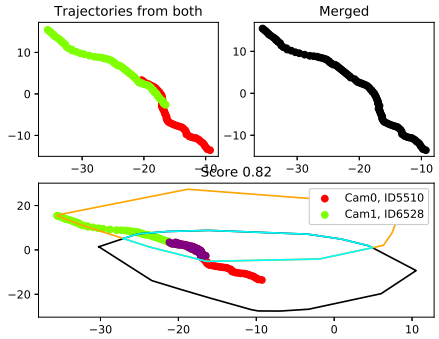
$\circledast$	P^{109}	P^{110}	P^{142}
P^{109}	409	150	188
P^{110}	115	357	72
P^{142}	139	153	497

where $P^k \circledast P^l$ is number of trajectories in $P^{k,l}$ for camera k.

Alg 1 was able to identify pair of trajectory with high similarity score for the different combination of cameras with examples from camera 109 and 110,

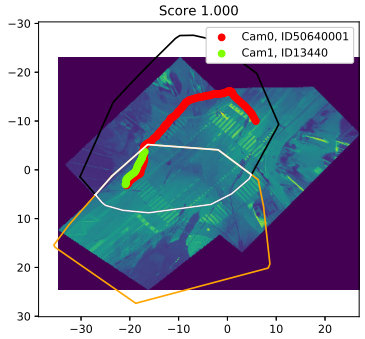


(a) A map overview of the intersection with adjusted x and y - axis over Mellingen.

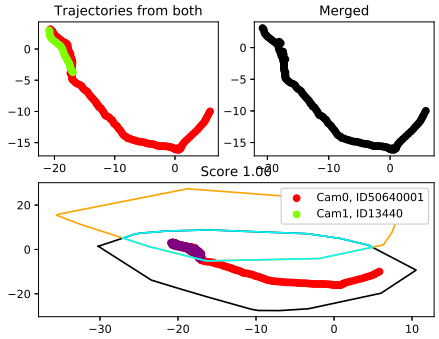


(b) Before (upper left) and after (upper right) merging with overview in regular x and y - axis.

Figure 4.36: Comparison between camera 109 and 110 with trajectory 5510 (red) from camera 109 and trajectory 6528 (green) from camera 110 with similarity score 1.



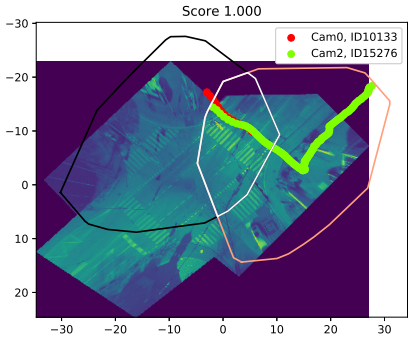
(a) A map overview of the intersection with adjusted x and y - axis over Mellingen.



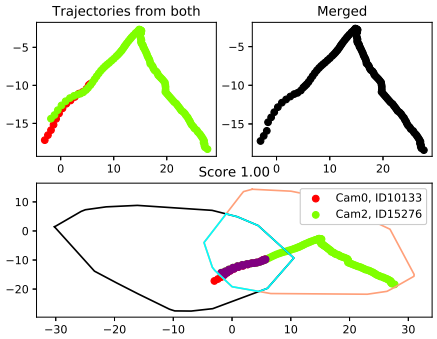
(b) Before (upper left) and after (upper right) merging with overview in regular x and y - axis.

Figure 4.37: Comparison between camera 109 and 110 with trajectory 50640001 (red) from camera 109 and trajectory 13440 (green) from camera 110 with similarity score 1.

camera 109 and 142 and

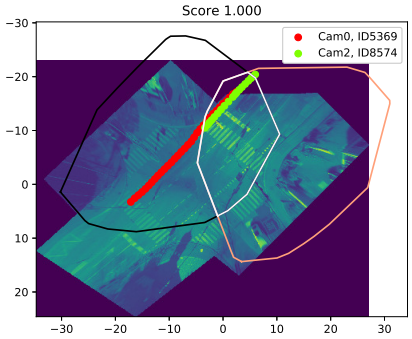


(a) A map overview of the intersection with adjusted x and y - axis over Mellingen.

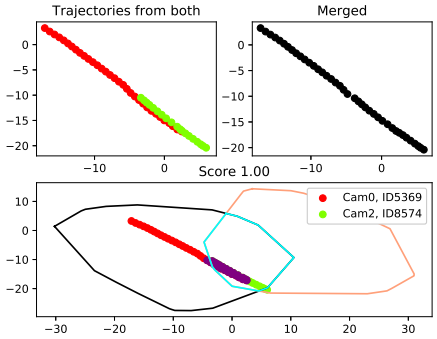


(b) Before (upper left) and after (upper right) merging with overview in regular x and y - axis.

Figure 4.38: Comparison between camera 109 and 142 with trajectory 10133 (red) from camera 109 and trajectory 15276 (green) from camera 142 with similarity score 1.



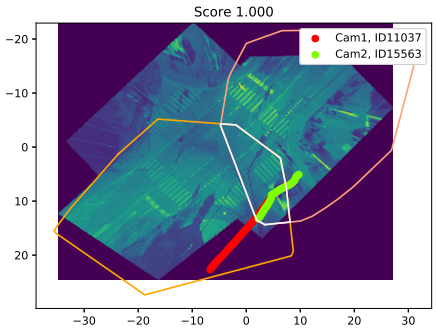
(a) A map overview of the intersection with adjusted x and y - axis over Mellingen.



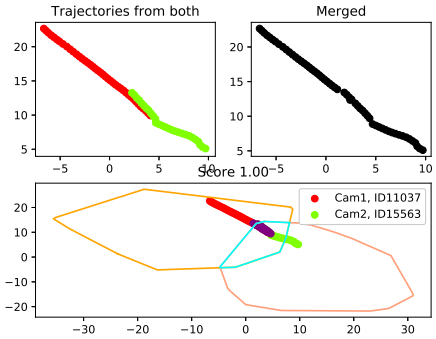
(b) Before (upper left) and after (upper right) merging with overview in regular x and y - axis.

Figure 4.39: Comparison between camera 109 and 142 with trajectory 5369 (red) from camera 109 and trajectory 8574 (green) from camera 142 with similarity score 1.

lastly 110 and 142.

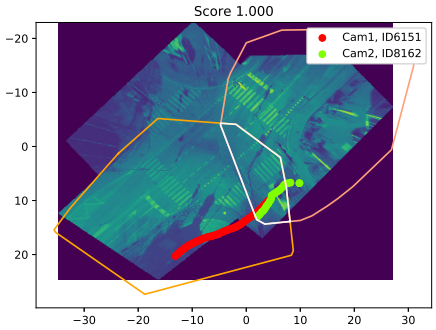


(a) A map overview of the intersection with adjusted x and y - axis over Mellingen.

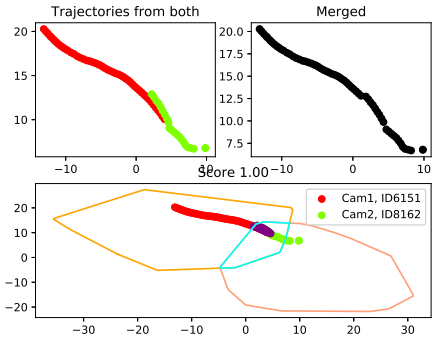


(b) Before (upper left) and after (upper right) merging with overview in regular x and y - axis.

Figure 4.40: Comparison between camera 110 and 142 with trajectory 11037 (red) from camera 110 and trajectory 15563 (green) with similarity score 1.



(a) A map overview of the intersection with adjusted x and y - axis over Mellingen.



(b) Before (upper left) and after (upper right) merging with overview in regular x and y - axis.

Figure 4.41: Comparison between camera 110 and 142 with trajectory 6151 (red) from camera 110 and trajectory 8162 (green) with similarity score 1.

Duplicates between the merging order occured when $\varphi_{142,109,110}$ was executed

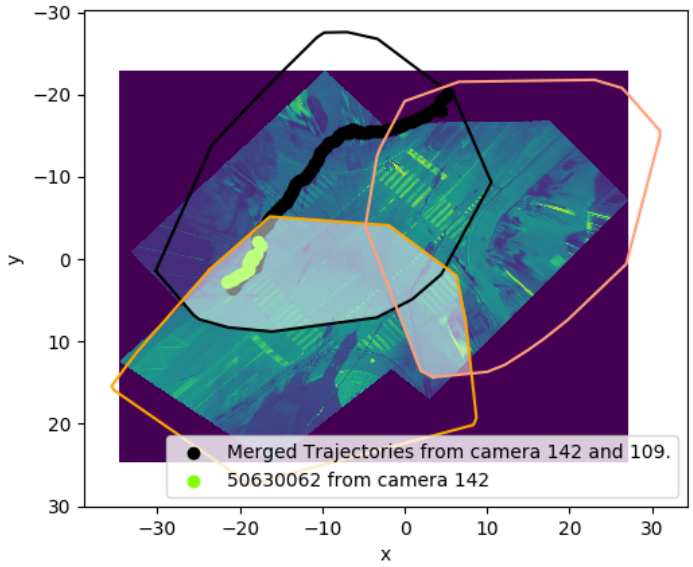


Figure 4.42: Merging between camera 142 and camera 109 (black) and the detection of a trajectory in camera 110 (green).

where the black trajectory was merged first and the green one got detected after. The green trajectory was however intercepted during the merging order $\varphi_{110,109,142}$

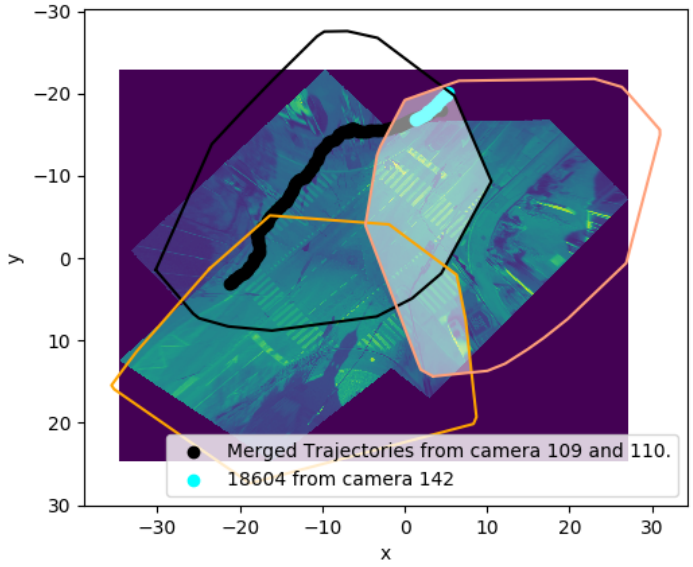


Figure 4.43: Merging between camera 110 and camera 109 (black) and the detection of a trajectory in camera 142 (cyan).

where in this case, the cyan one that was merged previously in fig(4.42). Both of which will be replaced by a total merge of all three trajectories.

4.3.2.2 Pedestrians

For pedestrians at Mellingen, the data looked like the following Cayley table:

Table 4.5: Cayley table of number of pedestrians with atleast ξ points in each polygon intersection.

$\circledast$	P^{109}	P^{110}	P^{142}
P^{109}	530	194	271
P^{110}	177	511	157
P^{142}	195	69	527

with the polygons

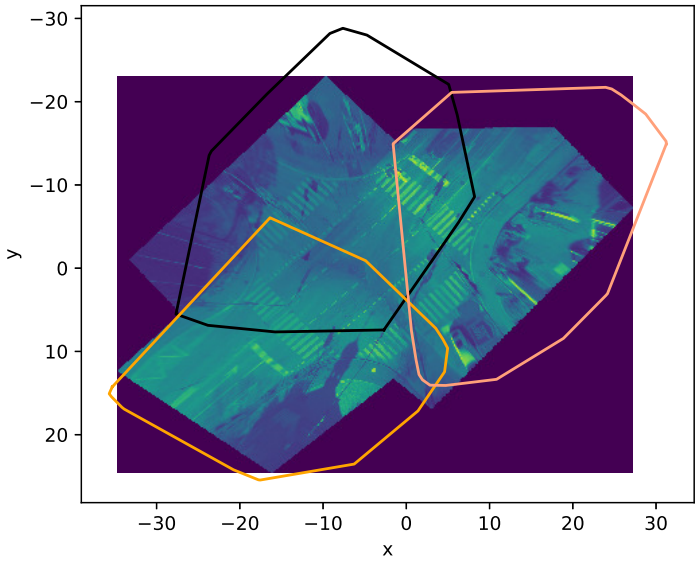


Figure 4.44: The polygons created by only using the data of pedestrians. Black is camera 109, orange is camera 110 and pink is camera 142.

Notice that in the case of pedestrians, $\bar{P} = \emptyset$, which means that the total amount of merging permutation can be reduced. The merging order of camera 110, camera 142 and camera 109 can for instance be discarded. Alg 2 did also work for combining two cameras and compare with the third. First, choose trajectories from camera 109 and 110 with a $\bar{\eta} \geq 0.8$ the results became

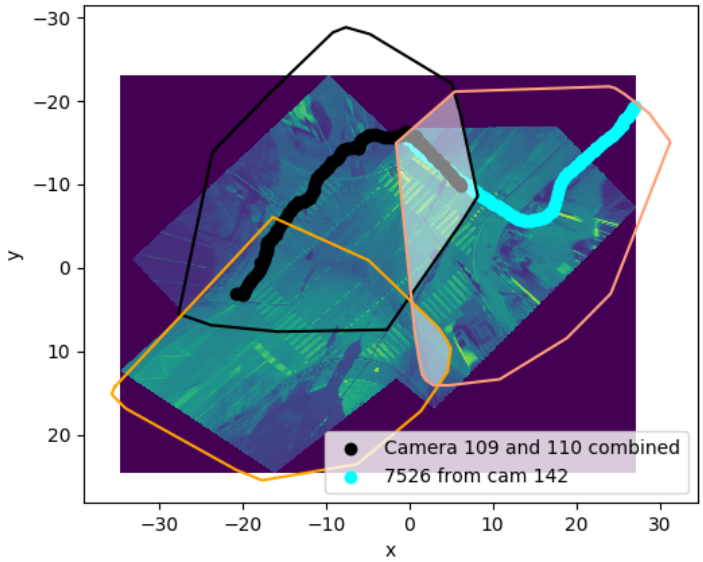


Figure 4.45: The polygons and the merged trajectories (black) and the soon to merged trajectory from camera 142. Black is camera 109, orange is camera 110 and pink is camera 142 and the white area is $P^{(109,110)\cap 142}$.

where the black trajectory $\tilde{\theta}$ is $\Theta(4655, 50400080)$ with respect to $P^{109\cap 110}$. The white area is $P^{(109\cup 110)\cap 142}$ after comparison with trajectories from camera 142. ID 7526 from camera 142 was found to be the one most appropriate one, with $\eta_{\varphi,7526} = 0.888$. The final merging $\Theta(\tilde{\theta}, 7526)$ became

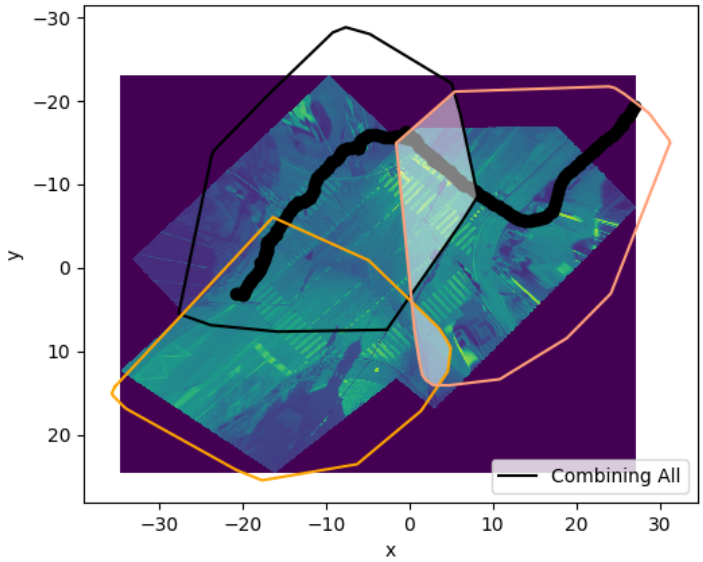


Figure 4.46: Total merging of all chosen trajectories (black).

In comparison to the chosen ID's original paths

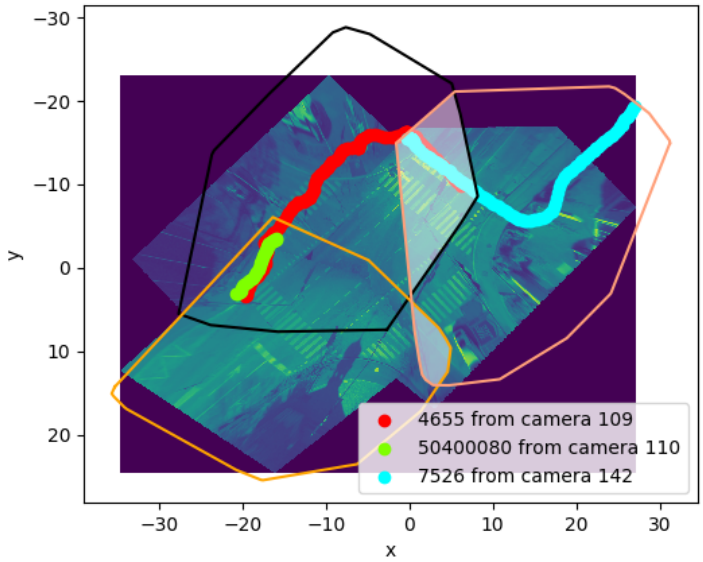


Figure 4.47: The merged trajectories seperately, green: trajectory from 110, red: trajectory from 109, cyan: trajectory from camera 142 with respect to $P^{(109,110)\bigcap 142}$ (white).

Another pairing of trajectory, but this time, first finding trajectories from camera 142 and camera 109 and then merge them. The original trajectories with the obtained polygons, became

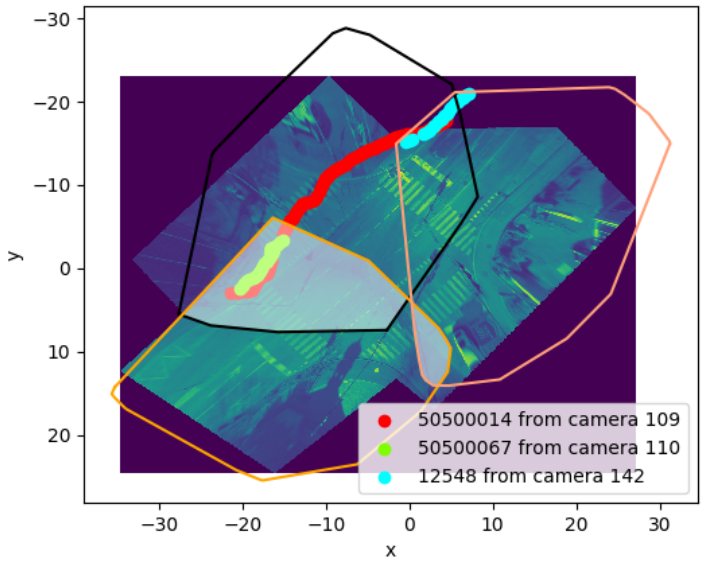


Figure 4.48: The merged trajectories seperately, green: trajectory from 110, red: trajectory from 109, cyan: trajectory from camera 142 with respect to white: $P^{142,110)\bigcap 109}$.

where the first merging step (i.e merging from camera 142 and 109) lead to

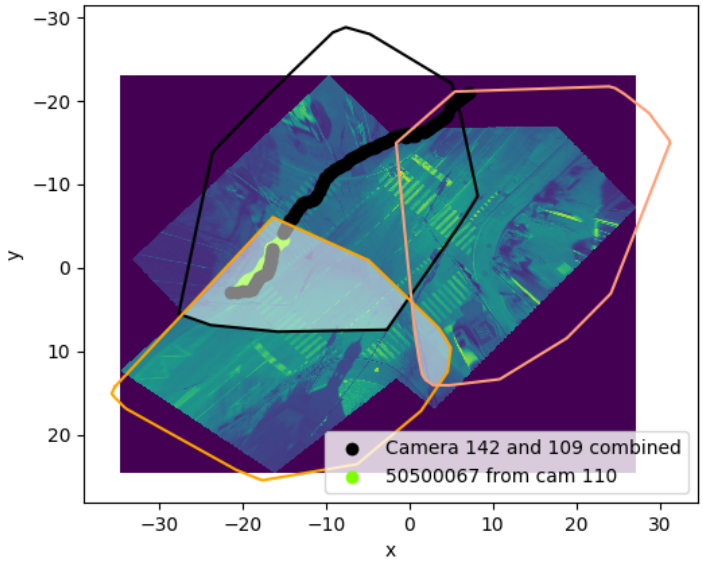


Figure 4.49: First merging step. Black: Merging of trajectories from camera 142 and 109, green: trajectory from camera 110, white: $P^{(142,109)} \bigcap^{110}$.

with the final merge being

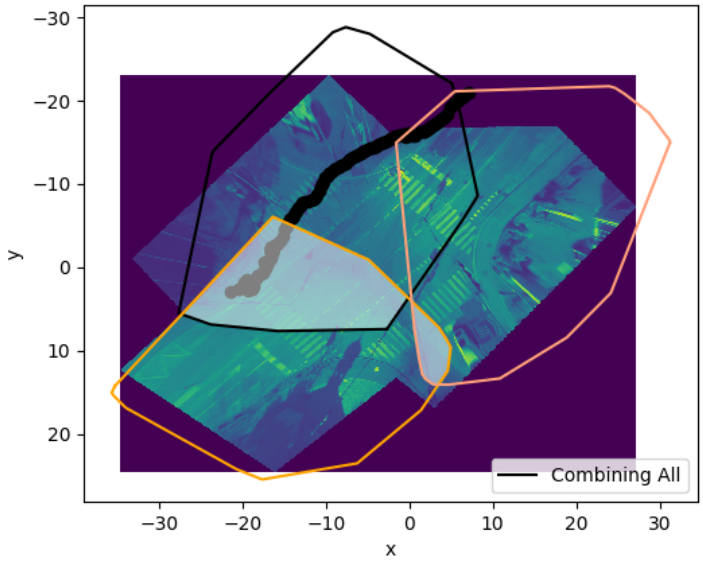


Figure 4.50: Total merging of chosen trajectories from all cameras.

where the final merging score was $\eta(\Theta(50500014, 12548), 50500067) = 0.76$.

4.3.2.3 Cars

The polygons for cars are

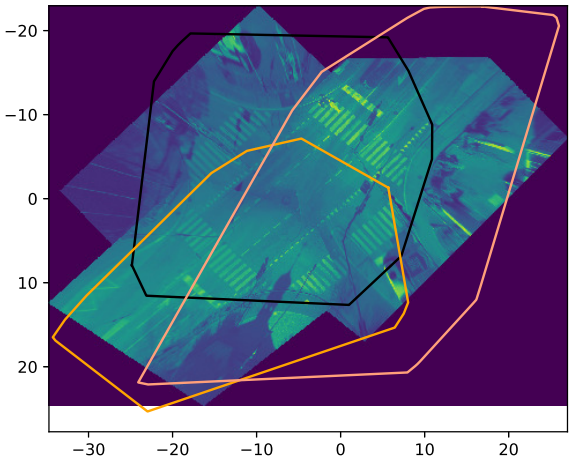


Figure 4.51: The polygons created by only using the data of pedestrians. Black is camera 109, orange is camera 110 and pink is camera 142.

where $\bar{P}$ is quite large and does contain large amount of data. The order is thus of greater importance since the paired trajectories

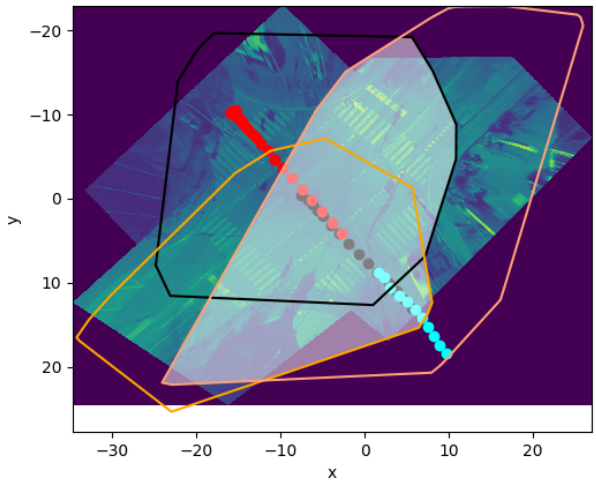


Figure 4.52: Paired trajectories was found in the order of 142 (cyan), 110 (black) and 109 (red). White: $P^{(142,110)} \bigcap {}^{109}$.

could only be found by the merging order of 110, 142 and then 109 since $P^{(110,142)\cap 109}$ (white) does cover enough of the trajectory from camera 109 (red) whereas the merging order 109, 110 and then 142 would result in

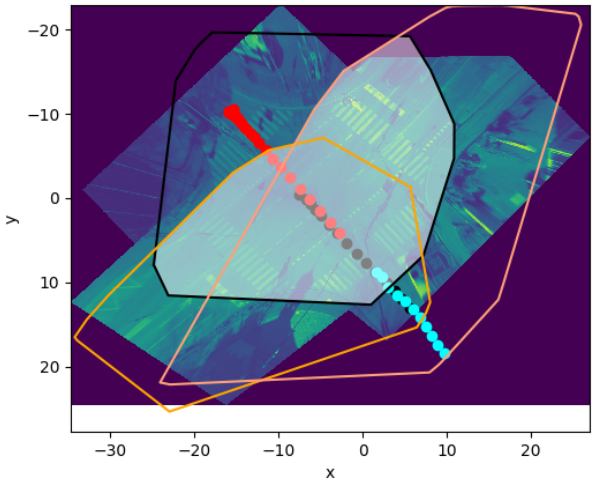


Figure 4.53: Paired trajectories was found in the order of 109 (red), 110 (black) was found. Not 142 (cyan). White: $P^{(109,110)\bigcap 142}$.

where cyan did not appear since $P^{(109,110)\cap 142}$ (white) does not cover enough points from trajectory in camera 142 (cyan).

4.3.2.4 Heavy Vehicles

The amount of heavy vehicles was rather low

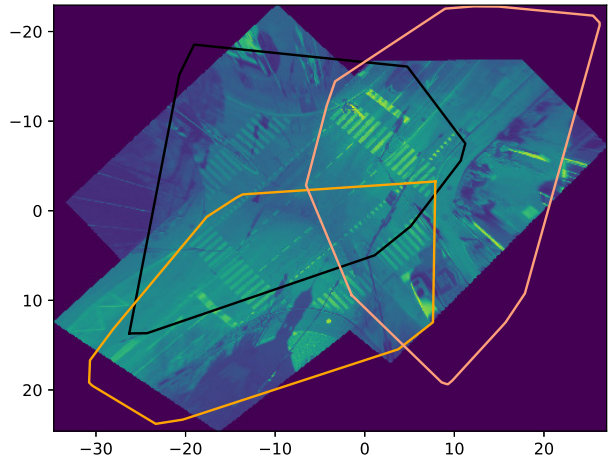


Figure 4.54: The polygons created by only using the data of pedestrians. Black is camera 109, orange is camera 110 and pink is camera 142.

but it produced decent polygons. The small amount of heavy vehicles could, in this traffic scen, have avail it in forming its polygons, compared to the polygons obtained by cars fig(4.51).

4.3.2.5 Overall

The merging threshold for both steps during a two step merging process was chosen as in Skövde, i.e $\bar{\eta}_1 = 0.35$. The results can be presented as

Table 4.6: Number of merged trajectories for $\bar{\eta} = 0.35$. $(\pm n)$ determines the amount of duplicates.

	Pedestrians	Cyclists	Cars	Heavy vehicles	Total
$P^{109 \cap 110}$	64	16	3153	28	3261
$P^{109 \cap 142}$	55	12	7105	28	7200
$P^{110 \cap 142}$	25	22	72	0	119
$P^{(109,110) \cap 142}$	0 (+7)	0 (+1)	38 (+6)	1	39
$P^{(109,142) \cap 110}$	7 (−7)	1 (−1)	408 (−6)	1	417
$P^{(110,142) \cap 109}$	0	0	11	0	11
Total	151	51	10787	60	22096

where the parameters (ϵ, T) was chosen as $(1, 1)$ and $(\pm n)$ determines the duplicates that were found during another permutation where $(+n)$ determines which data set they are added at and $(-n)$ where they are removed from.

5

Discussion

5.1 Camera alignment

Being able to align the coordinate systems of different cameras using a data driven method such as the method we have used would save time and effort, since one does not need to manually find keypoints in the camera pictures to match by hand.

The main problem in finding the rigid transformation between the coordinate systems of the different cameras seems to be that having a large number of points that lie outside the region where the cameras' views overlap (and other types of "noise") will most likely lead to most point set registration algorithms failing to find the correct solution. Since which points belong to the overlapping region is not known beforehand, one of the main things to do in order to succeed in finding the rigid transformation is to reduce the amount of noises attempt to make sure that at least a substantial number of points lie in the overlapping region. Another problem related to this is if the overlapping region is very small or narrow. In this case there may not be enough points in the overlapping region.

Another point is that information that is independent of the coordinate systems can possibly help make point set registration algorithms more accurate, and at the very least speed up computations. For example, in our case we used the timestamps associated with each point in our point set registration algorithm. By doing this a large number of point correspondences can immediately be excluded, i.e. if a the timestamps of the two points in the different cameras differ by some amount, then they should not be matched together (which in our case is accomplished by downweighing the corresponding matching probability). The inclusion of other coordinate independent data may be able to improve the accuracy and speed of the point set registration, e.g. colors of the traffic objects corresponding would allow one to remove or downweigh correspondences between traffic objects with different colors (unfortunately, the cameras used in this project are not color cameras).

5.2 Matching Trajectories

As of its own, the LCSS algorithm Alg(4) is capapble of finding paired trajectories with high probability of being the same trajectories. However, for the paired trajectories with some ambivalence (paired trajectories around $\bar{\eta} = 0.35$) should be scrutinized with a complementary algorithm. In addition, the parameters of (ϵ, T) was done by intuition rather than some metric. A possibly more sensible approach would to take the average distance and time between each traffic object in each

intersection and finding an optimal (ϵ, T) before performing LCSS. The merging threshold for the first and second merging should in this case, be the same since only one of the two trajectories is capable of matching with the third which is the same as taken the matching one and match it with the same third one. The version presented on LCSS, namely the similarity matrix H^{kl}, is a sence, robust against noises since noises tends to behave "unnateral", thus making it harder to pair them with other real trajectories. Nonetheless, if most or many noises behave the same, Alg(2) would not be able to detect them. As a method to reduce noise (especially for two cameras), for other algorithms such as TimeCPD, then it seems reliable enough.

As of merging, it should be accentuated that the merging method (2.11) should only be used as a temporary merge when there are more than two cameras. The aim of a temporary merge is to preserve the original two trajectories without too much loss of information before comparing with the other available cameras. For the final merging (regardless of the number of cameras) when all the comparison is done, the presented merging method is only applicable if the matching occur at the start of one trajectory while also being matched at the end of the other trajectory, such as Figure(4.36). In other instances, it is still incomplete Figure(4.17). The final merging, which in addition to (2.11), should consist of removal of unwanted and redundant points for the "natural" appearance of a trajectory, after comparing all local cameras.

LCSS has been used in camera alignment but is not necessarily bounded by it since it can be applied in any type of data set consisting of trajectories with respect to some attributes. These examples could be from microscopic levels in the movement of some bacteria to military related applications in radar related systems. In the case of data analyzis, LCSS have shown to be capable to reduce noise for other algorithms such as TimeCPD. In addition, LCSS has also been shown to be accessible to implement as LCSS have been used in this thesis both as coordinate and velocity for finding similarities.

6
Conclusion

LCSS is capable of reducing noise for other algorithm such as TimeCPD, however the area of the overlap is of importance. The bigger the overlap the more robust it gets for obtaining a transformation. The trade off of having a big overlap is the information collected from cameras is being reduced. In addition to the area of the overlap, the general movement inside the overlap is also of importance since the overlap can still be large but the majority of the points moves in a small area of the overlap. Even though the amount of permutations scale pretty fast when merging trajectories, each intersection is unique with an inclusion of flow of the given traffic scen, one should be able to reduce the number of permutations. LCSS and the merging algorithm presented is an important step and is capable of pairing the divided trajectories within each camera but will still need some work to do.

Bibliography

[1] L. Lee, R. Romano, and G. Stein, "Monitoring activities from multiple video streams: Establishing a common coordinate frame," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 758–767, 2000.

[2] B. Eckart, K. Kim, and K. Jan, "Eoe: Expected overlap estimation over unstructured point cloud data," in *2018 International Conference on 3D Vision (3DV)*. IEEE, 2018, pp. 747–755.

[3] X. Zhu, J. Liu, J. Wang, W. Fu, H. Lu, and Y. Fang, "Global trajectory construction across multi-cameras via graph matching," in *2011 Sixth International Conference on Image and Graphics*. IEEE, 2011, pp. 801–806.

[4] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*. John Wiley & Sons, 2019, vol. 793.

[5] A. Myronenko and X. Song, "On the closed-form solution of the rotation matrix arising in computer vision problems," *arXiv preprint arXiv:0904.1613*, 2009.

[6] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Proceedings 18th international conference on data engineering*. IEEE, 2002, pp. 673–684.

[7] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.