CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Dead reckoning using road networks

Master's thesis in Computer science and engineering

TIM KINDERBY
CARL RIDDERSTOLPE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Dead reckoning using road networks

TIM KINDERBY
CARL RIDDERSTOLPE

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Dead reckoning using road networks

TIM KINDERBY
CARL RIDDERSTOLPE

Supervisor: Carl-Johan Seger, Computer Science and Engineering
Advisor: Jonas Jalminger, Alkit Communications
Examiner: Nir Peterman, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Dead reckoning using road networks

Tim Kinderby
Carl Ridderstolpe
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Accurate vehicle position tracking in environments with limited Global Positioning System (GPS) availability remains a significant challenge in navigation systems. This thesis presents a dead reckoning algorithm that fuses data from accelerometers, gyroscopes, and speed sensors using a Kalman filter, supplemented by map-based corrections. By integrating known road network constraints and implementing a scoring system based on dynamic time warping, the algorithm enhances positioning accuracy during GPS outages. Extensive benchmarking across various road types and outage durations demonstrates strong performance on highways and country roads, with up to 77% accurate route prediction in short outages. The system also includes a backtracking mechanism to correct mispredictions, although limitations in scoring and prediction drift remain. Results suggest that context-aware parameter tuning and improved scoring methods could further enhance robustness in urban environments and extended outages.

Keywords: Kalman filter, computer science, dead reckoning, road network.

Acknowledgements

We want to thank our supervisor Carl-Johan Seger for giving great guidance within the thesis, algorithm and work structure that made the thesis what it is.

We also want to thank Jonas Jalminger and Jonathan Olsson at Alkit for providing ideas, driving data and a lot of technical help.

Lastly we want to thank Alkit for the resources, opportunity and data that made this thesis possible together with the fruit, coffee and fika giving us the energy to do the work.

Tim Kinderby and Carl Ridderstolpe, Gothenburg, 2025-06-30

Contents

List of Figures

List of Tables

1

Introduction

The ability to track a vehicle's position is enabled by the Global Positioning System (GPS). However, as noted by Kao et al. [1], GPS reliability can falter in environments with limited satellite visibility, such as areas with a high density of tall buildings, tunnels, or densely forested areas. Such disruptions may cause complete loss of position information leading to the inability to navigate due to the only known position being the last. To address this limitation, a method or algorithm that estimates the current position based on the last position is needed. Algorithms that do this position estimation are called dead reckoning algorithms and are often used as a method for estimating a vehicles position in the absence of GPS.

Dead reckoning algorithms estimate current position based on the last known position and sensor data such as velocity, acceleration, and orientation. The fusion of data from multiple sensors is often performed using Kalman filtering. However, without any external correction, such approaches can accumulate error over time. To mitigate this, map-matching strategies using known road networks can be employed to constrain the estimated trajectory and reduce drift.

This thesis introduces a dead reckoning algorithm that combines data from multiple sensors, such as accelerometers, gyroscopes, and speedometers, with the structure of the road network to improve positioning accuracy during GPS outages. The algorithm builds upon a Kalman filtering framework for sensor fusion and incorporates map-based corrections to enhance robustness and reduce accumulated error. By integrating road geometry into the filtering process, the proposed method can provide more accurate and stable position estimates than traditional dead reckoning approaches.

1.1 Background

There have been many studies on the concept of Dead reckoning with varying methods[2]–[10]. For dead reckoning and studies on it, vehicle data is needed. One method for collecting vehicle data is the Wireless Communication Unit (WCU) from Alkit Communications. The WCU is able to collect all available vehicle data but also has its own accelerometer which is always available. Calculating a position from accelerometer data is researched within the field of inertial navigation and in papers such as [3] and [4]. In the articles, the authors describe how one could estimate position from accelerometer data. Even though there are growing errors, the techniques

can form a good base for Dead Reckoning algorithms using units such as the WCU.

However, the best results are achieved when multiple inputs are used. The most common method uses Kalman Filtering. One of the earlier papers discussing this method is [5]. It feeds data from an accelerometer, gyroscopes, and tilt sensors, into a Kalman Filter. A recent systematic review found that Kalman filtering remains a widely used algorithm for sensor fusion [11]. These findings hint that Kalman filtering is likely a suitable approach for the project.

Another way to increase accuracy and reduce accumulative error is to incorporate map-based corrections using the road network as in [2]. They estimate the velocity and yaw rate of the vehicle with a particle filter instead of a Kalman filter. The data they use come from the vehicles Anti-lock Braking System (ABS) and the Electronic Stability Program. They then treat the road network as a graph and use the yaw data to identify the most probable road segment.

The article [12] also uses road networks for better performance. Their system employs an extended Kalman filter to predict vehicle movement using data from ABS wheel sensors, and applies Belief Theory to select the most likely road segment for correction. They report strong performance in a 2 kilometer test case, showing that the method can operate without GPS for extended periods when properly initialized.

While conceptually similar in its use of dead reckoning and map correction, the approach in this thesis differs in several important ways. Instead of evaluating individual map segments using geometric and directional heuristics, our system queries a routing engine (GraphHopper) to generate short forward paths from the current estimated position. These paths are scored against the predicted motion to determine the best fit, and a backtracking mechanism is employed when confidence is low. The scoring also incorporates an alignment step to get even better results. Furthermore, rather than relying on ABS-based odometry, a motion model based on vehicle velocity and yaw is used. This algorithm is systematically evaluated across a large dataset with varied GPS dropout durations, reflecting a different but complementary research focus.

1.2 Objectives

The primary objective of this thesis is to design, implement, and evaluate a dead reckoning algorithm that integrates vehicle sensor data with road network information. Specifically, the project aims to:

- Develop a Kalman filter-based sensor fusion system for estimating vehicle position during GPS outages.
- Integrate a map correction component using road network data.
- Design a scoring and alignment method to evaluate predicted and candidate paths.
- Evaluate the algorithm's performance under various driving conditions and outage durations through benchmarking.

1.3 Limitations

The thesis focuses on developing and evaluating a dead reckoning system using a Kalman filter and map-based correction. However, several limitations shaped the scope of the thesis:

- **Limited data sources**: Data was only collected from one vehicle, driving in one country. This limitations was due to resource limitations, the data available was limited to a single country and vehicle.
- **Evaluation scope**: The benchmark methodology simulates complete GPS outages but does not account for real-world degradations such as noisy or drifting GPS signals. Incorporating such conditions could provide a more detailed assessment of the algorithm's reliability. The reason for this limitations was due to time constraints, and due to the duration of the benchmarking itself.
- **Road network assumptions**: The algorithm assumes that the vehicle remains on mapped roads at all times. Scenarios like off-road driving, parking lots, or private roads not represented in OpenStreetMap therefore leads to incorrect predictions or uncorrectable deviations. This was considered to be outside the scope of this thesis, since a big part of the objective is to evaluate the use of road networks themselves.
- **Static parameters**: The algorithm uses fixed parameters for route distance and sliding window size. Due to time constraints dynamic parameters weren't implemented.
- **Hardware assumptions**: All processing was conducted offline on a desktop-class processor. This was excluded early on, since the scope of the thesis was the algorithm itself, not a hardware implementation.

2

Theory

This chapter presents the theoretical foundation for dead reckoning and its key components, including sensor models, Kalman filtering, and map-based corrections using OpenStreetMap and GraphHopper.

2.1 Dead reckoning

Dead reckoning is a navigation technique used to estimate the current position of a moving object by integrating motion data over time, it is described in detail in [13]. The core idea is to begin from a known location and continuously update the position estimate based on velocity, heading, and elapsed time. It is commonly used in situations where external positioning systems (like GPS) are unavailable or unreliable, such as in tunnels, cities with dense road networks, or indoor environments.

In principle, dead reckoning applies to any mobile system as long as some form of motion data is available. In this thesis, the focus is on vehicular dead reckoning using velocity and yaw (heading) measurements. These are typically derived from onboard sensors such as wheel speed encoders, inertial measurement units, or GPS-derived motion estimates.

In the context of vehicle navigation, dead reckoning involves these key steps:

1. **Initialization**: The process begins with a known GPS position before signal loss. Initial heading can be calculated from multiple prior GPS points or obtained via a compass or inertial sensors.

2. **Heading Estimation**: Heading is typically updated using sensors such as gyroscopes that measure angular velocity, or inferred from for example wheel speed and steering input.

3. **Speed Estimation**: Speed is estimated using data from sensors such as the vehicle's speedometer, wheel odometry, or via integration of accelerometer readings.

4. **Position Estimation**: The system integrates the heading and speed estimates to calculate the new position, often using a filtering method such as a Kalman filter to improve accuracy and reduce noise.

2.2 Sensors

Sensors are the foundation of dead reckoning systems, providing real-time measurements of motion and orientation. The following are the primary sensor types used in this thesis.

2.2.1 Accelerometer

Accelerometers measure linear acceleration along three axes (X, Y, and Z), relative to free fall. Velocity $v(t)$ is obtained by integrating acceleration $a(t)$ over time t:

$$v(t) = v(t_0) + \int_{t_0}^{t} a(\tau)d\tau$$

However, accelerometers suffer from noise, bias, and drift. These errors are exacerbated when integrated, leading to significant inaccuracies in position estimation unless corrected by other sensors or external references.

2.2.2 Speedometer

The speedometer provides direct velocity measurements. These are used both to validate and correct velocity estimates derived from accelerometer integration, offering greater robustness during GPS loss.

2.2.3 Gyroscope

Gyroscopes measure angular velocity, typically around three orthogonal axes. Orientation is obtained by integrating angular velocity:

$$\theta(t) = \theta(t_0) + \int_{t_0}^{t} \omega(\tau)d\tau$$

Where $\theta(t)$ is orientation angle and $\omega(t)$ is angular velocity. These measurements are critical for estimating heading changes, which significantly influence position calculations in dead reckoning.

2.3 Kalman filter

Kalman filtering is a recursive mathematical algorithm that provides optimal state estimation for linear dynamic systems from a series of noisy measurements. It was first introduced by Rudolf E. Kalman[14] and there have been articles written on how a Kalman filter works [15].

The filter assumes that both the system's state and the observations of the state have Gaussian distribution, characterized by a mean and covariance that describe the possible values a state can have. Then, it predicts the next state's mean and covariance based on the current state and the system's linear model. After which

it will correct its prediction based on the prediction's mean and covariance together with an observation and its covariance. With each prediction and observation the filter tracks the certainty of the prediction and gains better estimates of the system. It does this by recursively calculating the weighting between the prediction and observation at each step.

2.3.1 State modelling

In more detail, the Kalman filter models a linear system as:

$$x_{t+1} = Ax_t + b + v$$

Where x_t is the state vector at time t representing the true state that the filter is trying to estimate. A is the matrix representing the transition of the system, b is the control vector which consists of constants and external effects on the system, and v is the vector representing the Gaussian noise in the system.

Because the filter assumes the system follows a Gaussian distribution $\mathcal{N}$, the state can be expressed as:

$$X_t = \mathcal{N}(x_t, Q)$$

where Q is the covariance matrix representing the noise of the system.

A simple one dimensional example of this in dead reckoning is a drag racing car driving along a straight road. That would be represented by a state x_t consisting of the position p_t in meters and the speed v_t in meters per second, state transition matrix A derived from the equation $x_{t+1} = x_t + \Delta t \cdot v_t$, b representing the accelerating force of the car with value a, and the noise v which normally one would get by observing the system, but in this example it is assumed to be 0.1 for both position and speed. The system can then be described as the following:

$$x_{t+1} = \begin{bmatrix} p_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \begin{bmatrix} 0 \\ a\Delta t \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

2.3.2 Prediction

Since the true state is not directly observable and the filter only tries to estimate it based on the expected value, the filter needs to predict the state. The prediction is derived from calculating the expected value of the true state which removes the noise since it has a Gaussian distribution. The prediction $\hat{x}$ of the system therefore becomes:

$$E[x_{t+1}] = E[Ax_t] + E[b] + E[v] = A\hat{x}_t + b + 0 = A\hat{x}_t + b = \hat{x}_{t+1}$$

However, it is important to quantify the uncertainty of the prediction. Therefore Kalman filters track the uncertainty of the prediction through the covariance matrix P_t representing the covariance between all states (including the variance of the states in the diagonal). The covariance matrix is calculated by taking the variance of the system:

$$Var[x_{t+1}] = Var[Ax_t] + Var[b] + Var[v] = A^2P_t + 0 + Q = AP_tA^T + Q = P_{t+1}$$

Again using the fact that this is Gaussian we can write the prediction as:

$$\hat{X}_{t+1} = \mathcal{N}(\hat{x}_{t+1}, P_{t+1})$$

In the drag racing car example from above, the structure remains largely unchanged, except for the definition of Q. For the sake of simplicity it is assumed that the system is independent and Q is then given as a diagonal matrix with the values of v in its diagonal. In a real world application, since position is dependent on speed it follows that the variance of the position is dependant on the variance of the speed. Even if the covariances are calculated from observations they are often tuned to get better results[16]. This means that setting covariances representing the real world is not often prioritized since tuning Q for optimal results is preferred.

2.3.3 Update

This represents the predictions the Kalman filter makes. However, observations about the system are often available. These observations, represented by the vector z_t, are often provided by some sensor or other system that has its own noise that can be assumed to be Gaussian. The noise is represented by the vector d making the observation be given by the equation:

$$z_t = x_t + d$$

Since the observation is Gaussian as well it can be written in the form of:

$$Z_t = \mathcal{N}(x_t, R)$$

Where R represents the covariance matrix of the observation.

In the example of a drag racing car, an observation of the system could be a signal from the car's odometer and speedometer. Measurement devices tend to have some noise but still give valuable information about the state, especially since the Kalman filter takes this noise into consideration. And as with the system's covariance Q, R can be assumed to be independent and be defined as a diagonal matrix with the values of d in its diagonal. This is because just as Q, R is tuned for optimal results in real world applications.

To get the best estimation of the system, observations of the system will also be used. This is done by linearly combining the prediction and observation. To do this, a matrix K_t representing the linearization factor, which is also known as the Kalman gain, is introduced. The linearization is represented as the following equation:

$$\hat{x}_t^{joint} = \hat{x}_t(I - K_t) + z_tK_t$$

What makes the Kalman filter a good estimation tool is that it at each observation tries to estimate the best Kalman gain and refine the estimate. By looking at the joint distribution of the prediction and observation, the Kalman gain can be computed. Since both the prediction and observation have a Gaussian distribution, they can be joined to get a combined estimate of the system much like the linearization. The new joint estimate $\hat{X}_t^{joint}$ becomes:

$$\hat{X}_t^{joint} = \hat{X}_t \times Z_t = \mathcal{N}(\hat{x}_t, P_t) \times \mathcal{N}(z_t, R) = \mathcal{N}\left(\frac{\hat{x}_t R + z_t P_t}{P_t + R}, \frac{P_t R}{P_t + R}\right) = \mathcal{N}(x_t^{joint}, P_t^{joint})$$

By rewriting the prediction of the joint distribution to the form of the linearization, the Kalman gain is derived. This is done as follows:

$$x_t^{joint} = \frac{\hat{x}_t R + z_t P_t}{P_t + R} = \hat{x}_t \frac{R}{P_t + R} + z_t \frac{P_t}{P_t + R} = \hat{x}_t \left(\frac{P_t + R}{P_t + R} - \frac{R}{P_t + R}\right) + z_t \frac{P_t}{P_t + R} =$$

$$\hat{x}_t \left(1 - \frac{P_t}{P_t + R}\right) + z_t \frac{P_t}{P_t + R} = \hat{x}_t (1 - K_t) + z_t K_t \implies K_t = \frac{P_t}{P_t + R}$$

With the joint distribution, it is also possible to derive the new covariance P_t^{joint} given when joining together the prediction and observation. The new covariance is given as:

$$P_t^{joint} = \frac{P_t R}{P_t + R}$$

The step of predicting the state based on its model and joining that prediction with an observation of the system are the two main steps of the Kalman filter called prediction and update. The power of the Kalman filter comes from its recursive structure of predictions and updates where it balances how the system is modelled to work and how it is observed to work.

3

Methods

This chapter describes the implementation of the dead reckoning algorithm developed in this thesis. The system combines a custom Kalman filter with a map correction module that leverages known road network data to constrain the predicted trajectory during GPS outages. The chapter also discusses data collection and the technologies used to develop and evaluate the system.

3.1 Tools And Technologies

A stack of tools and technologies, both existing and custom, was necessary for the algorithm's implementation, including data processing and visualization.

3.1.1 OpenStreetMap

OpenStreetMap (OSM) is a collaborative, crowd-sourced geographic database that models the world as a graph of:

1. **Nodes**: Point features with latitude and longitude coordinates that represent specific points in space.
2. **Ways**: Represent linear features such as roads, paths, and boundaries as ordered lists of nodes.
3. **Relations**: Defines how different elements relate to each other.

In road networks, nodes can represent things like intersections, traffic lights, or points where the road geometry changes significantly.

The model forms a connectivity where road segments (ways) connect to a shared node. Intersections are represented by nodes shared by multiple ways. This structure allows for route calculation.

3.1.2 GraphHopper

GraphHopper is an open-source routing Java library. Once integrated into a project it provides a way to access road network data. By default it uses OpenStreetMap for road network data. It also provides several functionalities helpful for a dead reckoning algorithm:

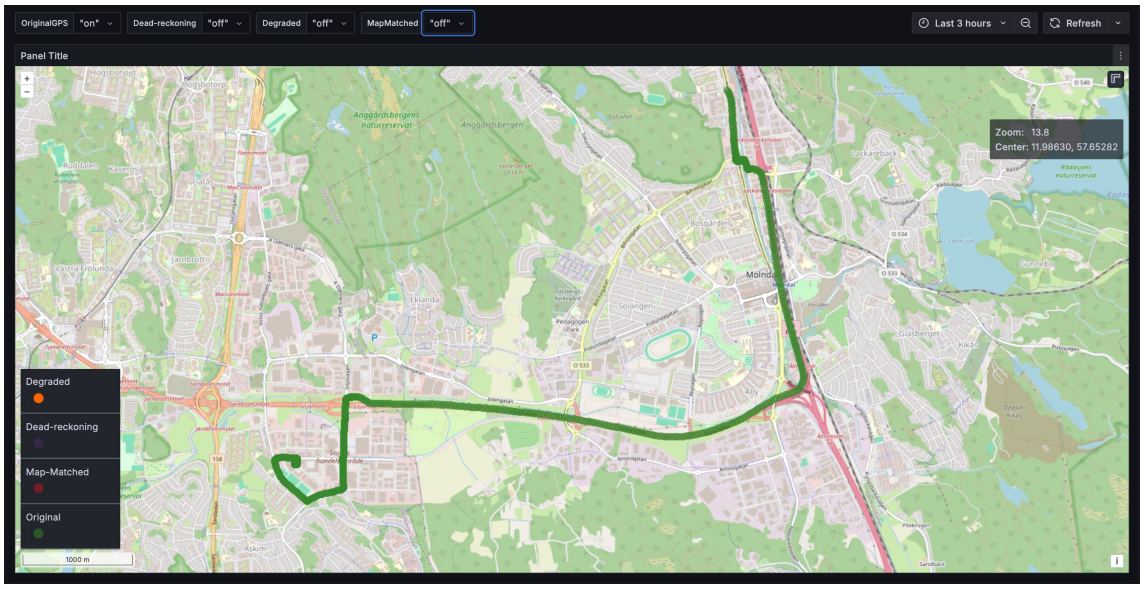

Figure 3.1: The Grafana interface

- **Routing**: Calculates the fastest or shortest path between multiple points with support for different transportation modes.
- **Snap to Road**: Maps arbitrary GPS coordinates to the nearest point on the road network.
- **Isochrone Calculation**: Generates accessibility maps showing areas reachable within specified time or distance thresholds.

3.1.3 Visualization - Grafana

The open-source visualization software Grafana is used to help intuitively visualize the algorithms output. Grafana was chosen because of its data visualization capabilities and its customizable dashboards. Grafana interfaces with a time-series database, InfluxDB, which serves as its data source.

A custom tool was developed that processes sensor data through a proprietary library that reads the particular data file. Then the sensor readings and the output of the algorithm are sent to InfluxDB via its REST API. Grafana then queries InfluxDB to generate real-time visualizations on its dashboard, enabling intuitive monitoring of the results of the algorithm.

3.1.4 Accelerometer re-orientation

The orientation of the WCU accelerometer is not known since the installation of the WCU varies depending on the vehicle. This means to get usable information from the accelerometer the orientation of it is needed. To get the orientation of the WCU a calibration step has been implemented. This step takes data from the cars accelerometer which is mounted in a known manner (such that X is longitudinal, Y is lateral and Z i vertical) and the WCUs accelerometer data for the same time

period. With this data a cross covariance matrix is computed on which a Singular Value Decomposition[17] is applied to get the rotation matrix. The rotation matrix can then be used to correct all past and future WCU accelerometer readings for this vehicle.

This approach enables the development and validation of the dead reckoning system under diverse scenarios, while avoiding the unpredictability and logistical complexity of live field testing.

3.2 Data Collection

The primary data source used in this thesis originates from a vehicle equipped with a Wireless Communication Unit (WCU). This hardware platform is capable of collecting various types of sensor data during normal vehicle operation. The WCU is integrated with the vehicles internal systems and continuously records key signals relevant for dead reckoning, such as acceleration, yaw rate, speed, and GPS position.

The data is accessed via WICE, a proprietary platform developed by Alkit Communications. WICE provides interfaces for querying and exporting recorded datasets.

To simulate real-world conditions and test the robustness of the algorithm during GPS outages, the collected data can be artificially degraded. This is accomplished by selectively removing GPS measurements from the dataset. These "virtual outages" allow for controlled evaluation of the algorithms ability to maintain accurate position estimates over varying outage durations, road types, and driving conditions.

3.3 Kalman Filter Implementation

A custom Kalman filter is implemented to estimate the vehicle state by fusing data from multiple sensors. The implementation is designed to estimate the position of a vehicle using data from an accelerometer and select vehicle signals. While additional sensor data could be incorporated, this thesis focuses only on the data sources outlined here. The implementation follows the Kalman filter theory presented in 2.3.

3.3.1 State representation

The filter is configured to track a five-dimensional state vector consisting of position coordinates (x, y) in meters, speed in meters per second, yaw angle in radians, and yaw rate in radians per second. The origin (0,0) of the coordinate system is defined by the first available GPS position. Figure 3.2 shows how position and yaw is defined for the state model.

$$x = [pos_x, pos_y, speed, yaw, yawRate]^T$$

Acceleration is handled as a control input to the system, driving changes to the vehicle's speed and orientation.

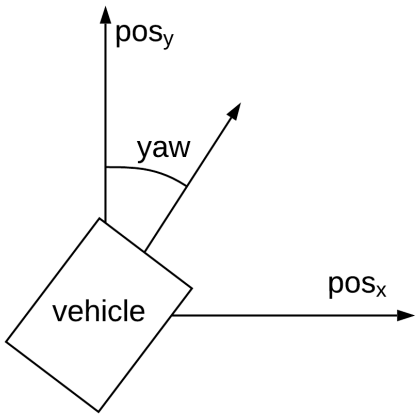

Figure 3.2: Illustration showing how position and yaw is defined.

3.3.2 Prediction Step

The prediction step advances the state estimate based on a motion model and control input. In the implementation, the filter uses accelerometer readings as inputs.

$$speed_{k+1} = speed_k + accX \cdot dt$$

$$yaw_{k+1} = yaw_k + yawRate_k \cdot dt$$

$$pos_{x,k+1} = pos_{x,k} + speed_k \cdot sin(yaw_k) \cdot dt$$

$$pos_{y,k+1} = pos_{y,k} + speed_k \cdot cos(yaw_k) \cdot dt$$

The speed is updated using the current speed and the integration of the incoming accelerometer data. The yaw is updated using the current yaw and the integration of the yaw rate. The positions along both x- and y-axis are updated with the last position and the integration of the speed in the direction of the current yaw angle.

3.3.3 Update Step

The filter accommodates multiple sensor updates, each with a dedicated update method. These include:

- **GPS Updates**: Provide position corrections and, if a previous GPS point is recent enough, also a yaw correction. The yaw is derived from the angle between the most recent consecutive GPS coordinates.
- **Speed Sensor Updates**: Corrects the speed estimate.
- **Yaw Updates**: Corrects the yaw value.
- **Yaw Rate Sensor Updates**: Corrects the angular velocity.

Each update applies the standard Kalman filter update equations:

$$y = z - Hx$$

$$S = HPH^T + R$$
$$K = PH^T S^{-1}$$
$$x_{new} = x + Ky$$
$$P_{new} = (I - KH)P$$

Where:

- z is the measurement vector,
- H is the measurement matrix (configured differently for each sensor type to only account for that sensor),
- R is the measurement noise covariance which is different for each sensor,
- K is the Kalman gain,
- P is the state error covariance matrix.

This structure ensures that each sensor update only affects the relevant part of the state vector and respects the specific uncertainty associated with that sensor.

3.3.4 Linearization of the process model

Since the state transition equations involve trigonometric functions (due to yaw-based motion), the process model is non-linear. To accommodate this, the state transition needs to be linearized at each prediction step. The transition matrix F is constructed with the following elements:

$$F = \begin{bmatrix} 1 & 0 & sin(yaw) \cdot dt & 0 & 0 \\ 0 & 1 & cos(yaw) \cdot dt & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.5 Initialization

The Kalman filter requires an initial estimate of the vehicle's state and its associated uncertainty. These are used to define the initial conditions as follows:

- **Position**: Set to the coordinates of the first GPS measurement.
- **Yaw**: Calculated as the angle between the two first GPS points.
- **Speed**: Estimated as the distance between the two GPS positions divided by the time elapsed.
- **Yaw Rate**: Initialized to zero, assuming the vehicle begins with no angular motion.

The system and all of the signals each have their own covariance matrix that are initialized. The values for the matrices was found by observing the covariances in historic driving data and fine tuning them for better predictions.

3.4 Map correction algorithm

In most real-world scenarios, it is reasonable to assume that vehicles drive on predefined roadways. This assumption significantly constrains the possible positions and movements of a vehicle, as it eliminates large regions of the map that are physically inaccessible. By leveraging this constraint, dead reckoning accuracy can be significantly improved, particularly during GPS signal outages, by limiting drift and correcting potential errors through the use of map data.

This section describes how the developed algorithm utilizes road network information to enhance dead reckoning, detailing the logic of the main algorithm loop, error correction through backtracking, route generation, and scoring methods.

3.4.1 Main algorithm loop

Initially, the algorithm is run during periods with available GPS and vehicle sensor data. This phase serves to calibrate the Kalman filter, allowing it to converge toward an accurate state estimate, and to fine-tune its internal covariance matrices and Kalman gains.

Once the GPS signal is lost and position updates cease, the system enters its predictive phase. At this point, the current vehicle state and the internal Kalman filter state are saved as a node n in a linked list. This list maintains a history of all past filter states and predictions, enabling future corrections through backtracking if needed.

From this point, the system performs the following steps:

1. **Prediction phase**: The Kalman filter is run forward for a predefined distance, denoted as *predictDistance*, generating a predicted path.

2. **Candidate path generation**: Using the same starting point, the algorithm calculates all points that the vehicle could feasibly reach within *predictDistance* plus an aditional window size. Routes to these points are generated using a routing function.

3. **Route scoring**: Each candidate route is scored based on its similarity to the predicted path. Routes that score below a predefined threshold are discarded, while those meeting the threshold are stored in node n, along with the corresponding vehicle and Kalman filter state.

4. **Path selection and update**: The best-scoring route is assumed to represent the most probable trajectory of the vehicle. The predicted positions are then replaced with this route, and the Kalman filter is updated with the final position and yaw of the selected path. The new state is appended to the linked list, and the algorithm continues iteratively from this updated state.

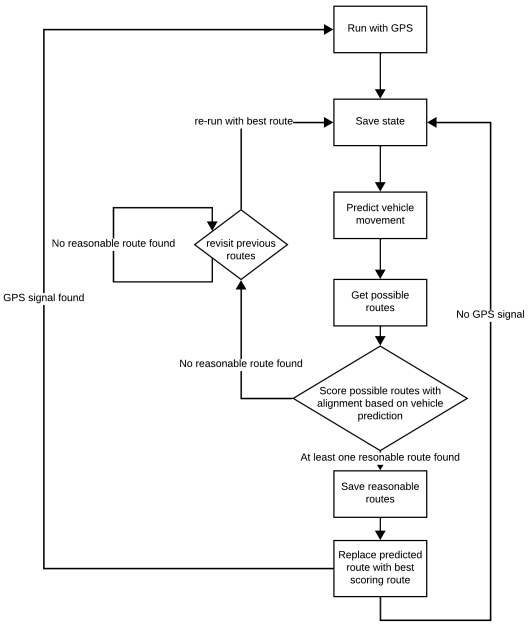

Figure 3.3: Flowchart showing how the map correction algorithm works

3.4.2 Error correction

Due to the presence of similar paths in road networks, the algorithm can occasionally make incorrect assumptions about the vehicle's trajectory. For instance, it may erroneously select a highway exit when the vehicle actually remains on the main road.

Such misidentifications may persist for several iterations if the incorrect route continues to align reasonable well with predictions. However, problems arise when none of the possible paths match the predictions. For example, the algorithm might expect the vehicle to enter a roundabout after the exit while in reality the vehicle continues on the highway.

When all potential paths receive scores above the threshold, the algorithm recognizes that there likely has been an error in previous iterations and initiates a backtracking procedure. It returns to the previous saved state and examines alternatives paths that scored well but weren't selected. If it identifies such a path, the algorithm discards the incorrect branch, adopt the alternative high-scoring path, and continues from that point.

If there are no paths scoring well enough the algorithm will backtrack further until it has found a potential path to run with instead. This ensures that even if an incorrect path was initially chosen, and not detected for multiple loops, the error will be detected and corrected.

In extreme cases the algorithm could backtrack all the way back to the root node and find no viable paths, resulting in failure. In practice this should be rare if the other parts are modelled correctly. With an accurate road network, reliable scoring

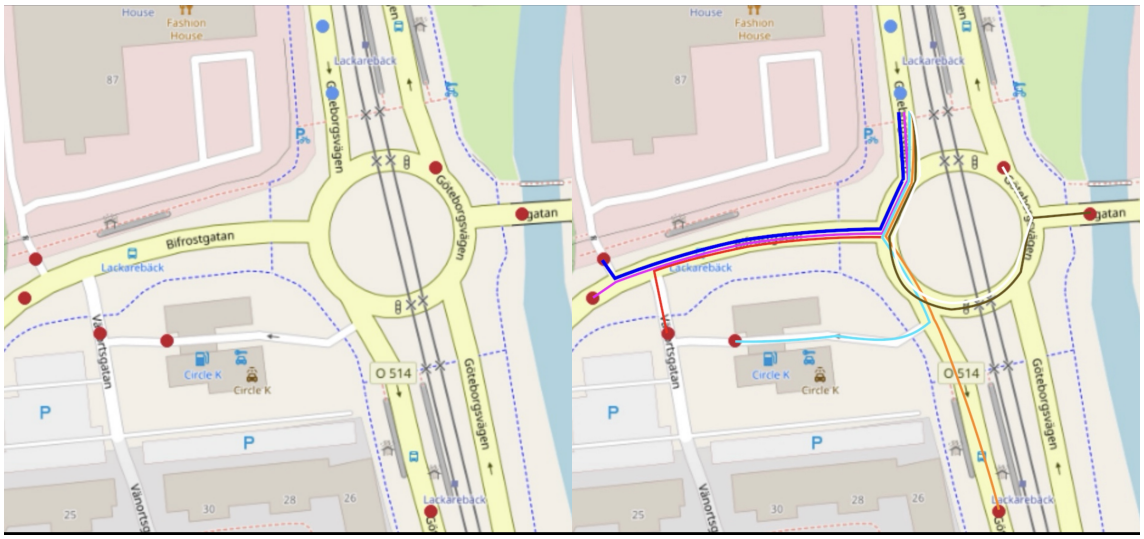

Figure 3.4: Illustrates isochrone and routing functions

metrics, and good predictions, the algorithm should successfully identify a valid path for the vehicle.

3.4.3 Calculating possible routes

Route generation is divided into two primary steps: isochrone generation and path routing.

Isochrone Generation: An isochrone defines all positions reachable from a given point within a specified distance or time constraint. This is calculated using GraphHoppers built-in isochrone function, which returns a list of all reachable points within the given constraint. The raw isochrone output includes all reachable points, but only the outermost ones are of interest. These are extracted by filtering out interior nodes, retaining only those at the boundary of the reachable region.

An implementation challenge arises with the isochrone implementation due to the relatively sparse road network, where nodes often don't precisely align with the constraint boundary. For example, when requesting an isochrone with a 100-meter constraint, the outermost point might only be 80 meters from the starting point. This occurs because the next available point in the road network, located 120 meters away, exceeds the constraint and is therefore excluded from the results. To address this, the following methodology is used:

1. Identify all edges that cross the constraint boundary.
2. For each crossing edge, calculate the precise interpolated point where the constraint is met.
3. Record the interpolated points, which collectively form the constraint perimeter.

After identifying all possible endpoints on the constraint perimeter, we calculate optimal routes from the starting point to each endpoint using GraphHopper's routing

Figure 3.5: Comparison of route alignment techniques showing how sliding window correction improves matching between predicted and actual routes.

functionality. Since the resulting paths are relatively sparse, they are enhanced by inserting additional points through interpolation. Figure 3.4 shows both the result of the isochrone step and routing step.

It's important to note that only optimal routes are computed rather than all possible routes. This simplification could potentially miss the actual route taken in some scenarios, particularly when vehicles make unexpected detours. This limitation becomes more significant as the constraint size increases and the number of possible route variations grow.

3.4.4 Time interpolation for routes

When receiving a route from GraphHopper the points in the route lack accurate timestamps. Since the position of a vehicle at a certain time is relevant and not only its route, timestamps need to be interpolated. This is done by going through all the points of a route from start to finish. The first point's timestamp is the start time of the route. Subsequent timestamps are given by taking the previous point's timestamp and adding the time it took the car to drive the segment between those points. The segment time is given by the distance of the segment divided by the speed of the car at the start of the segment. The time is given by the cars speed signal at the time of the start point.

3.4.5 Alignment

Misalignment can occur between predicted routes and actual results due to various factors. To address these mismatches, the algorithm applies a sliding window alignment technique. This method systematically shifts the predicted route along the different possible routes at specific intervals. At each interval step, the alignment is scored individually, and the optimal alignment with the lowest score is selected. This can lead to the algorithm doing unnecessary and wrong alignments on road segments with few changes in shape such as straight roads. To hinder this the score with no alignment is favoured by having its score lowered by 40%. This technique

compensates for spatial offsets and improves the accuracy of route comparison. Figure 3.5 shows how sliding the prediction can result in a better match against the road network route.

3.4.6 Scoring

To be able to choose the correct route, the different possible alternatives need to be scored. Below are all scoring methods explored and implemented. The final implementation of the algorithm uses Dynamic Time Warping for its scoring since it got the best results during development.

3.4.6.1 Geographical similarity

One method for scoring routes is by calculating how close the two routes are to each other geographically. Since both routes are represented by a list of geographical points the scoring can be given by calculating the minimum distance between the points in the prediction and the road network route.

To get the minimum distance between a point and a route, the closest point on each segment in the route has to be found. The following formula is applied for a point (x_p, y_p) to find the the nearest point (x, y) on a segment span by the points (x_0, y_0) and (x_1, y_1):

$$t \leq 0 => (x, y) = (x_0, y_0)$$

$$1 > t > 0 => x = x_0 + t * (x_1 - x_0), y = y_0 + t * (y_1 - y_0)$$

$$t \geq 1 => (x, y) = (x_1, y_1)$$

Where

$$t = \frac{(x_p - x_0)(x_1 - x_0) + (y_p - y_0)(y_1 - y_0)}{(x_1 - x_0)^2 (y_1 - y_0)^2}$$

With the closest point found for all segments it is simple to take the smallest euclidean distance between the predicted point and the closest points on the segments.

$$d = \sqrt{(x - x_p)^2 + (y - y_p)^2}$$

The score s is then given by the root mean squared (RMS) of the minimum distances for all predicted points.

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^{n} d_i}$$

Where n is the amount of predicted points and d_i is the minimum distance for point i. RMS was chosen since it acts as a weighted average that puts more weight on

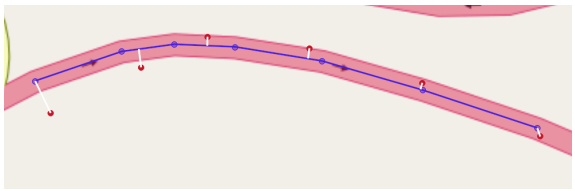

Figure 3.6: Example of distance calculation between prediction (red) and road network route (blue)

higher values. This means that large errors are punished more severely and will gain higher scores making routes with large errors be less likely to be chosen.

Figure 3.6 shows an example of the distance calculation between a predicted route and a road network route where the white lines represent the calculated distance.

3.4.6.2 Yaw similarity

Another method to measure the similarity of routes are their shape, which can be measured in the difference in yaw. To get a score of the yaw similarity, the difference in yaw between two corresponding segments are measured.

First the two routes are split into segments of equal length defined by the segments between the points in the predicted route. This is done by modifying the road network route and interpolating points in it to make it match the length of the segments in the predicted route. The yaw of each segment is then calculated as:

$$y_j = arctan\left(\frac{(p_{i+1,long} - p_{i,long})p_{i+1,lat}}{p_{i,lat} * p_{i+1,lat}(1 - (p_{i+1,long} - p_{i,long}))}\right)$$

where y_j is the yaw of segment j in radians, $p_{i,long}$ is the longitude of point i in radians and $p_{i,lat}$ is the latitude of point i in radians.

Afterwards the score is given by the least means square error calculated with the following equation:

$$s = \sqrt{\frac{1}{n}\sum_{j=1}^{n-1} y_{j+1} - y_j}$$

where s is the score and n is the amount of segments in a route.

3.4.6.3 Dynamic time warping

Dynamic time warping (DTW) is another method used to measure similarity between two sequences. In this projects implementation the algorithm is applied as follows:

1. **Distance Matrix Calculation**: For each point p_i in the candidate trajectory, the Euclidean distance to every point q_j in the reference trajectory is calculated, creating an $m \times n$ distance matrix D where $D(i, j) = ||p_i - q_j||$.

2. **Accumulated Cost Matrix Computation**: A DTW matrix C is then constructed iteratively, representing the minimum accumulated cost of aligning the sequences up to each pair of points. This matrix is built using the relation: $C(i,j) = D(i,j) + \min\{C(i-1,j), C(i,j-1), C(i-1,j-1)\}$

The final DTW score, found in the bottom-right cell $C(m,n)$ of the accumulated cost matrix, represents the total alignment cost between the two sequences. Lower scores indicate greater similarity between the routes.

3.5 Benchmark method

To comprehensively evaluate the algorithm, it was benchmarked using real-word driving data which was edited to generate GPS outages of varying durations. These benchmarks will show in what environments the algorithm perform best in and which parameters it should have in different environments.

The benchmarks were performed on data collected from the personal vehicle of an Alkit employee, covering the period from March 7, 2025 to April 12, 2025. The dataset comprises 185 individual driving sessions, each lasting up to 30 minutes.

To evaluate the algorithms performance under various durations of GPS loss, we defined benchmark windows of fixed lengths:

30, 60, 90, 120, 180, 240, 300, and 600 seconds

Each benchmark window began with five GPS samples, followed by GPS deprivation for the remainder of the window. Because the data was sampled at a rate of one GPS point per second, this means the first five seconds of each run had valid GPS data, while the remaining $X - 5$ seconds were simulated GPS outages. Throughout the thesis, we refer to the full benchmark window length X as the *duration* of the run for simplicity, even though the actual GPS-deprived interval is slightly shorter.

For each file and each duration, multiple benchmark windows were generated. The number of windows per file was determined by dividing the total session length by the benchmark duration.

The algorithm was executed independently for each window, testing all combinations of the following parameters:

- **Route Distance**: 50, 100, 150, 200, 250, 300, 350, and 400 meters
- **Sliding Window Size**: 50, 100, 150, 200, 250, 300, 350, and 400 meters

These parameters influence the behaviour of the map correction algorithm and were varied to evaluate their impact on predictive accuracy.

Each run produced a predicted trajectory, which was evaluated against the original GPS route using Dynamic Time Warping (DTW) scoring. To normalize for different prediction lengths, the total DTW score was divided by the number of interpolated

points:

$$\text{Normalized Score} = \frac{\text{DTW Total Score}}{\text{Number of Interpolated Points}} \tag{3.1}$$

Interpolated points were placed at fixed distance intervals, ensuring comparability across benchmark durations.

Additionally, the distance between the final predicted point and the actual endpoint of the ground truth GPS trajectory was computed. This endpoint error serves as a secondary metric that highlights large-scale deviation or failure cases such as wrong turns, missed intersections, or total prediction route being too short or too long.

4

Results

This chapter presents the performance results of the dead reckoning algorithm developed in this thesis. A series of benchmarks were conducted to evaluate the algorithm under various GPS outage durations, driving conditions, and parameter configurations. The evaluation aims to assess in which scenarios the algorithm performs best, and how different parameter values affect accuracy and robustness.

4.1 Benchmark

To comprehensively evaluate the algorithm, it was benchmarked using real-word driving data which was edited to generate GPS outages of varying durations. These benchmarks will show in what environments the algorithm perform best in and which parameters it should have in different environments.

4.2 Benchmarking results

After running the benchmarking result, the data was gathered. Below is the presentation and visualization of the data together with data statistics and classification.

4.2.1 Benchmark classification of success thresholds

To interoperate the benchmark results, two classification threshold are used: one for route similarity, and one for endpoint deviation.

The theshold for the normalized DTW score is fixed at 30. This value was chosen based on manual inspection of a variety of results. Visual comparisons showed that a score below 30 generally indicates a strong alignment between the predicted route and the original GPS trajectory. This threshold is held constant across all run durations to maintain a consistent definition of what constitutes a good route match.

To account for the fact that longer prediction durations naturally lead to greater positional uncertainty, the acceptable endpoint deviation threshold varies depending on the run duration. The values were chosen based on reasonable drift expectations rather than being derived from a strict formula:

- 50 meters for 30s

- 100 meters for 60s
- 150 meters for 120s
- 200 meters for 180s
- 250 meters for 240s
- 300 meters for 300s
- 600 meters for 600s

These thresholds define the boundaries used in quadrant plots and in textual classification of benchmark quality. A result is considered Good Distance if the final predicted point lies within the specified threshold for that duration.

4.2.2 Quadrant analysis of benchmark results

A quadrant-based analysis was introduced to analyse the algorithms performance. The benchmark result was evaluated based on two metrics: the final normalized similarity score and the distance between the predicted and actual endpoints. These two axes were used to divide the data into four different performance categories:

- **Good Score, Good Distance (GG) (Green, lower left quadrant)**: These results indicate ideal predictions, both the route similarity and distance between endpoints are within acceptable thresholds.
- **Good Score, Bad Distance (GB) (Yellow, lower right quadrant)**: The algorithm predicted a route that matched the original trajectory well but ended too far from the true endpoint.
- **Bad Score, Good Distance (BG) (Purple, upper left quadrant)**: This indicates a route that terminates near the correct location but has poor similarity along the way.
- **Bad Score, Bad Distance (BB) (Red, upper right quadrant)**: These are failure cases where the predicted route both deviated significantly in shape and ended too far from the true location.

Each quadrant chart was generated for a specific outage duration, ranging from 30 seconds to 600 seconds.

Each duration has their own graph with figure 4.1 showing the results when duration was set to 30 seconds. The rest of the graphs are found in appendix A. The results are summarized in table 4.1.

4.2.3 Road type comparison

To give a better insight into how road environment influences the performance of the algorithm, quadrant analysis has also been segmented by road type: City, Country, and Highway. All plots have a fixed GPS outage duration of 30 seconds, allowing for comparison across different road types. The quadrants share the same definition

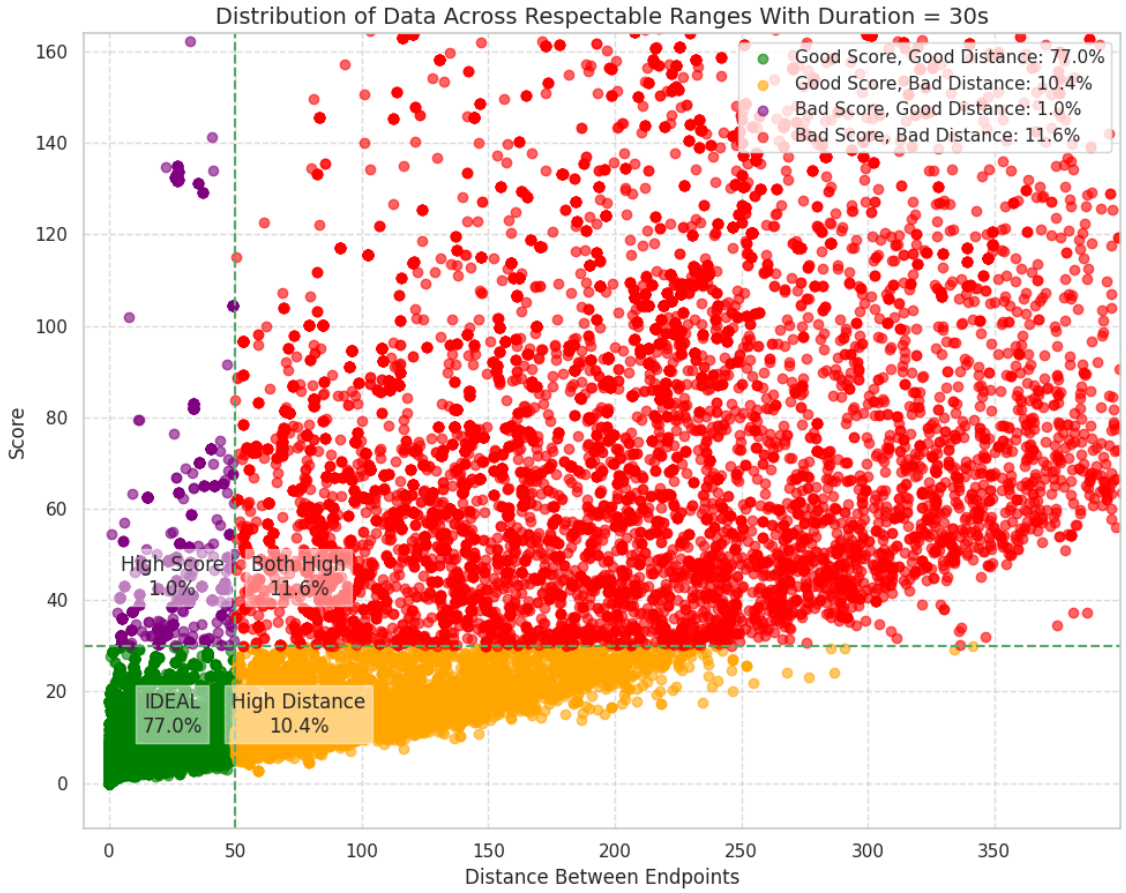

Figure 4.1: Quadrant distribution for 30 second GPS outage.

Table 4.1: Distribution of benchmark results across quadrants for each GPS outage duration

Duration (s)	GG (%)	GB (%)	BG (%)	BB (%)
30	77.0	12.5	3.1	7.4
60	73.3	8.2	2.3	16.1
120	60.9	11.4	3.6	24.1
180	56.6	11.0	2.7	29.6
240	49.0	9.7	3.0	38.3
300	44.9	8.4	3.9	42.8
600	32.1	4.1	2.1	61.8

as in section 4.2.2. The graphs showing the results are found in appendix B, with a summarization of the results found in table 4.2.

Table 4.2: Quadrant distribution of benchmark results for 30 second GPS outage, segmented by road type.

Road Type	GG (%)	GB (%)	BG (%)	BB
Highway	79.9	12.7	0.5	7.0
Country	75.8	11.4	1.1	11.6
City	61.1	4.4	3.5	31.0

4.2.4 Success rate by parameter configuration and road type

The following evaluation presents the percentage of successful predictions for various combinations of *routeDistance* and *slidingWindowSize*, segmented by road type. A predictions success is determined by the same metric as in section 4.2.2.

Success rates are reported for each pair of parameters across three environments: City, Country, and Highway. In appendix C, figures show the success rate of the three different environments as a heat-map with *routeDistance* on the vertical axis and *slidingWindowSize* on the horizontal axis. Table 4.3 summarizes the maximum success rate achieved for each road type and the parameter combination it used.

Table 4.3: Maximum success rate and corresponding parameters for each road type.

Road Type	Max Success Rate (%)	Route Distance	Window Size
City	72	100	50
Country	84	350–400	150–200
Highway	83	400	50–150

4.2.5 Full range of results

In figure 4.2 the full range of the results are shown. This includes data where the distance between endpoints reaches up to 4000 meters. The other figures have been zoomed in to make them more readable but figure 4.2 shows the distribution of results on the higher end of the spectrum.

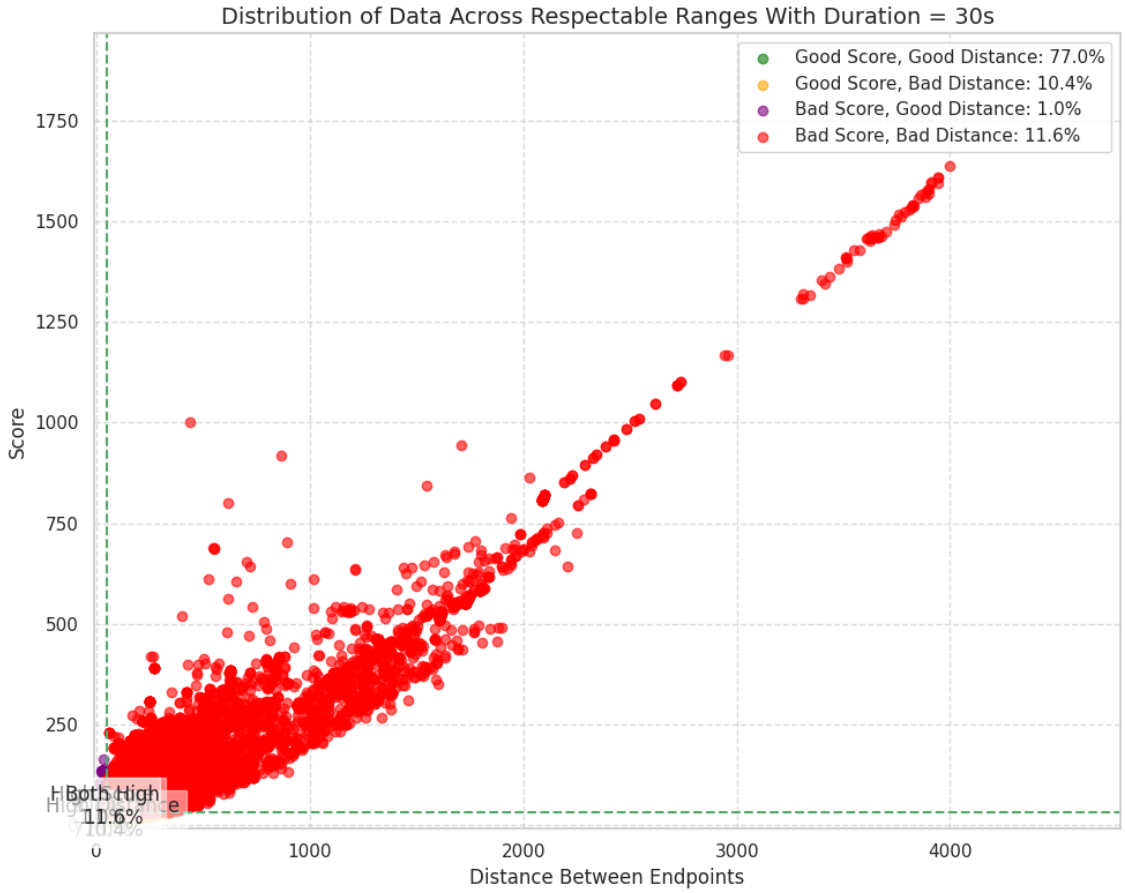

Figure 4.2: Graph showing the full range of results at duration = 30s

Figure 4.3: Showing how the algorithm chooses the wrong path

4.2.6 Recurring failure scenarios

While the benchmark result show promising performance some recurring failure scenarios were observed.

Choosing the wrong path: There are scenarios where there are two paths that both get a fairly low score even though one is clearly better then the other. As seen in figure 4.3, the algorithm chose to turn left even though the prediction does not show any turn. This is a drawback of using DTW scoring which purely calculates the cost of transforming the prediction path into the different possible routes.

Parking: When entering an area outside the road network the algorithm will still try to predict the vehicles movement even though the algorithm assumes vehicles to drive in road networks. If the car moves inside of this area the algorithm's prediction might move away from that area instead of standing still giving an incorrect estimate. This is showcased in figure 4.4

Accumalative distance error: During prolonged outages the algorithm can produce results where the prediction aligns well with the real route but is either too short or too far ahead. This can cause an error as shown in figure 4.5 where the turn has already happened. This will cause the algorithm to miss the turn and instead continue on.

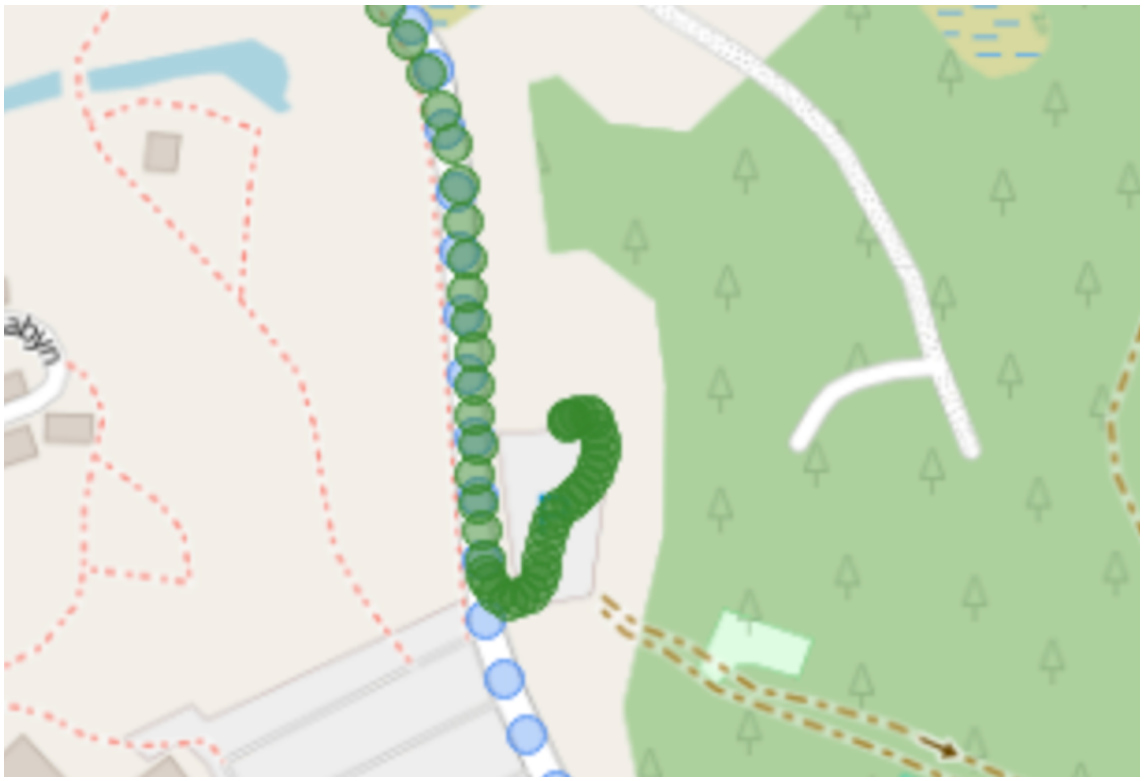

Figure 4.4: Showing parking error

Figure 4.5: Scenario where the prediction of the algorithm is too far ahead. The blue dots are the prediction and the green dots are the original GPS signal.

5

Conclusion

5.1 Discussion

The benchmarking results presented in Chapter 4 offers several insight into the performance characteristics of the adaptive dead reckoning algorithm under varying conditions. In this section the results of algorithm are analysed.

5.1.1 Trends in outage duration

One observation is the decline in performance as GPS outage duration increases. As seen in section 4.2.2, the proportion of runs classified in the "Good-Good" quadrant, which indicates both high route similarity and accurate endpoint prediction, drops significantly from 77.0% at 30 seconds to 32.1% at 600 seconds. This trend is expected, as error accumulation becomes increasingly difficult to mitigate without GPS corrections over longer intervals.

Notably, Good Score/Bad Distance results outnumber Bad Score/Good Distance, indicating that the route shape is often correct even if endpoint drift occurs.

5.1.2 Road type sensitivity

The segmentation of results by road type, seen in section 4.2.3, revealed a strong dependency on environmental complexity. Highways yielded the best performance (79.9% "Good score,Good distance"), followed by country roads (75.8%) and city roads (61.1%). This aligns with expectations: highways have simpler topologies and fewer turns, city roads have higher intersection density and more varying driving behaviour.

However, there is an additional factor causing the performance on city roads to be worse. In the data there exists the scenario of parking and driving in parking lots. This is something that the algorithm in its current form cannot properly handle. As seen in figure 4.4, the algorithm's reliance on the road network leads to it not allow the vehicle to leave the road. When this route is then scored it receives both a worse similarity score and a higher distance between the endpoints due to the vehicle driving outside the road network.

5.1.3 Optimal parameters

From the investigation of optimal parameter values certain configurations consistently yielded better results across different road types. Notably cities favour a low *slidingWindowSize*, this aligns with expectations as a larger *slidingWindowSize* in a dense road network will cause the algorithm to have more choices. The success rate for country roads differ where it generally favours combinations of both *routeDistance* and *slidingWindowSize* being larger. On highways, the parameters do not influence the performance as much although a slight improvement can be observed with an increased *routeDistance*.

These results support the intuition that parameter tuning should be context-sensitive and dynamic, which is further discussed in 5.2.5.

5.1.4 Systemic issues in prediction

When running the algorithm each prediction is not of perfect length and they are usually too short or too long. The sliding window in the map correction is supposed to counteract this but it does not always make an alignment or a correct alignment. An example is when a vehicle travels along a straight road, no alignment is typically made, and thus, the prediction error is not corrected. This can result in the prediction being too far ahead or too far behind. If the distance the algorithm is behind or ahead with grows it can result in the algorithm completely missing a turn or similar.

One such scenario is shown in figure 4.5 where the algorithm has predicted too far ahead and will predict that the vehicle is moving along the highway instead of taking the exit, when the vehicle took the exit. If the algorithm is very far ahead in its prediction like in figure 4.5 the error might not be fixed by backtracking since the prediction and the exit are not similar enough.

Another scenario is when the prediction is very far behind the actual position. If the algorithm is further behind than the size of the sliding window an alignment can not be made. This results in the error in length not being corrected and possible turns being missed by the algorithm. Figure 5.1 illustrates how even after sliding the path all the way to the end of the sliding window, it still does not reach the exit.

5.1.5 Outliers in score

As seen in section 4.2.5, there are times where the algorithm produces results which have an abnormal score and distance between endpoints.

From examining multiple runs with extreme scores it becomes quite clear that these are caused by the algorithm choosing the incorrect route once and not correcting itself. This means that if the algorithm makes a mistake once and takes a turn when it is not supposed to take one it can continue in that wrong direction for the rest of the run. If the incorrect route chosen is among the first predictions and forces the algorithm to estimate in an almost 180 degree different direction the score

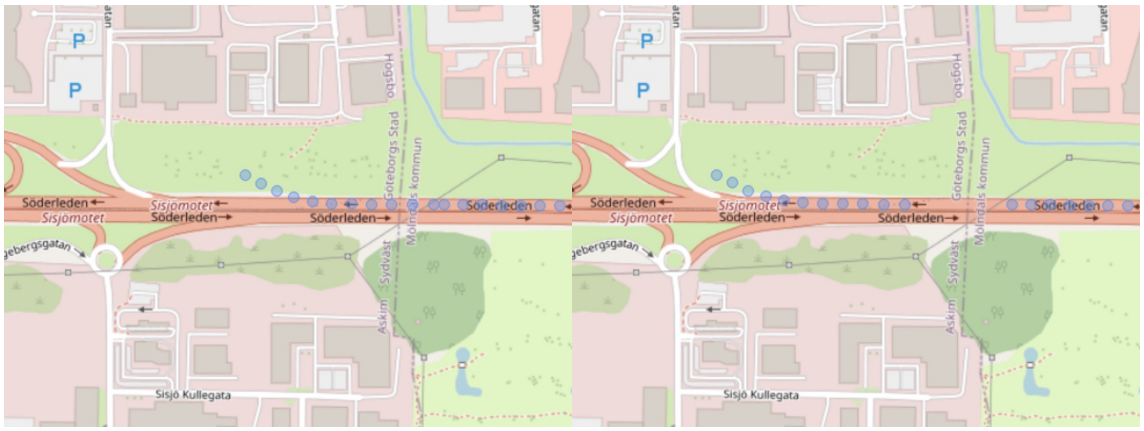

Figure 5.1: Shows the error that could occur with a too short sliding window

can become extremely high. While if the route is later in the run and only forces the algorithm to predict on a parallel street the score is not much higher than the interval of the majority of scores. The spread in scores is then explained by the timing and extremity of the incorrect route, while the extremely high scores are explained by the algorithm in practice predicting in the opposite direction of the actual route. The reason this is allowed to happen is because of the fact that it uses the road network to update the heading of the vehicle which is why the algorithm does not backtrack.

5.1.5.1 Suboptimal score threshold

One reason for the problem of the algorithm not correcting itself through backtracking could be a too high score threshold. When an incorrect route is selected, the algorithm should in theory encounter a scenario where there are no plausible routes since all routes have a too high score. The chances of two very similar routes existing from the same starting point should be quite low, especially when the length of the route increases. If the algorithm frequently accepts routes that are not very similar to the prediction, backtracking will not be done. To solve this the score threshold for accepting routes could be lowered, but that also increases the risk for correct routes not being accepted.

5.1.5.2 Problem in scoring

As discussed in 3.4.6 three different scoring methods were looked into and dynamic time warping (DTW) was chosen for the algorithm. DTW works by calculating the cost of transforming one route to another and showed to yield the best results, but all three have advantages and disadvantages. One problem in DTW is that it does not consider turning which is represented as changes in yaw. This causes problems where taking a turn on a country road can get the best score, which is illustrated in figure 4.3.

Although the algorithm includes a backtracking mechanism intended to recover from early route mispredictions, this feature did have some problems in practice. In

particular, incorrect routes were sometimes retained despite poor scoring, due to a combination of scoring imperfections and overly permissive thresholds. In several instances, backtracking failed to trigger even when the correct route was still within reach, suggesting bugs or instability in the rollback logic. To compensate for this, the score threshold was deliberately relaxed, allowing the algorithm to continue even with moderate confidence mismatches. While this improved stability, it also reduced the systems ability to reject clearly incorrect predictions. Addressing the limitations of both scoring and backtracking logic remains an important future improvement.

5.1.6 Real-time on-device computation

Although the benchmarking in this thesis was conducted entirely offline, preliminary observations suggest that real-time execution may be feasible. For instance, a 20-minute route is typically processed in just a few seconds on a Apple M2 processor. While embedded hardware used in in-vehicle systems is more constrained, it is unlikely to be so limited that the algorithm would exceed the actual driving duration. However, this is a simplified and indirect way of assessing performance. Real-time deployment would involve not only raw computation time but also considerations such as sensor input latencies, routing engine responsiveness, memory constraints, and the potential need for multithreading or prioritization. So while initial signs are promising, a more dedicated evaluation would be required to confirm practical feasibility.

5.2 Future works

There are many different directions that can be taken from this point. Following are some ideas that could be pursued. All of the proposed ideas are based on the current algorithm and its implementation. Some of the future works are also dependant on each other and could work well together but also conflict with each other.

5.2.1 Alternative benchmarking strategies

The benchmarking approach used in this thesis involved defining fixed-duration windows of simulated GPS loss, beginning with a short initialization period of five GPS points. While effective, several alternative benchmarking strategies could be explored in future work to more rigorously assess the algorithm's capabilities.

One alternative would be to benchmark based on specific scenarios or road features rather than time-based windows. For example, benchmarking performance during turns, roundabouts, merges, or complex intersections would highlight how well the algorithm handles edge cases.

Another alternative would be to have the algorithm operate with noisy and unreliable GPS data rather then a complete outage.

These benchmark strategies could lead to a more nuanced and comprehensive understanding of the algorithm's strengths and limitations under real-world conditions.

5.2.2 Improved initial yaw estimation

The algorithm currently assumes that the initial yaw estimate is sufficiently accurate, derived from the last available GPS readings. However, when GPS coverage is weak or the vehicle is moving slowly, these sources can be noisy or ambiguous. Future work could investigate more robust approaches to initial yaw estimation. Errors in yaw can have significant impact on early trajectory prediction, and a better approach could give a significant improvement to the overall performance of the algorithm.

5.2.3 More accurate GPS to local coordinates conversion

The conversion from GPS to local is fairly simplistic in the current implementation. It could be more accurate by using a more sophisticated algorithm. However the improvement might be negligible for all but very large distances.

5.2.4 Vehicle speed in scoring

As stated in 5.1.5.2 the scoring is not always perfect which increases the risk of the algorithm choosing the wrong route. One way to combat this is to take the vehicles speed into consideration. Since the vehicles speed is a part is the state estimate it could be used when scoring routes. If a route has a sharp turn but the prediction thinks the vehicle is driving at a speed of 90 km/h, it is probably not a plausible route and should get a high score. This is similar to the yaw scoring mentioned in 3.4.6 but instead compares the change in yaw to the estimated speed instead of comparing the route's and prediction's yaw.

5.2.5 Sliding window problem

As stated in 5.1.4 there could be problems with the algorithms prediction being too far ahead or too far behind. This can lead to grave errors and has been shown to create runs with extremely high score. Below are two different approaches to possibly combat the problem of prediction length in the algorithm.

5.2.5.1 Dynamic window size

An alternative approach is to dynamically adjust the length of the prediction itself. One strategy is to extend the prediction until the next intersection. A common failure scenario is when the prediction ends in the middle of a road segment, too far from the upcoming intersection, causing a missed alignment. By predicting up to every intersection, the likelihood of aligning to actual turns may improve.

However, this method does not fully eliminate the accumulation of prediction error. As with fixed-length predictions, no alignment is performed on straight roads. Thus, any error in prediction length continues to build until a turn is reached. Whether the prediction ends after a set time, distance, or at the next intersection, alignment gaps will persist during prolonged straight segments.

5.2.5.2 Dynamic prediction size

Another possibility is to instead have the length of the prediction to be dynamic. A proposed way to determine the prediction length is to predict until the next intersection. A scenario where the algorithm does not align to a turn is when the prediction ends in the middle of a road segment too far away from the intersection where a turn is happening. By instead predicting to every intersection the hope is to reduce missed alignments to turns made in intersections.

A problem that could remain with this method is the error of the prediction length could still be increasing. Since the algorithm does not align for straight roads when a car is driving on a straight road no alignment will be made. It does not matter if it predicts for a set amount of time, length, or until the next intersection if the road is straight and the car is driving straight on it. If the prediction then is too short or too long, it will not be corrected and the error in length will continue to propagate until an alignment is made.

5.2.6 Rerunning prediction with route

When running the map correction algorithm multiple times without incorporating new positional updates, the covariance of the Kalman filter's predictions gradually increases. This is because, without absolute measurements, the filter becomes less confident in its own estimates over time. As a result, the longer the correction loop runs without external updates, the greater the risk of a poor prediction potentially causing the algorithm to select an incorrect route or, in the worst case, discard the correct one.

To counteract this, positional updates are needed, and the selected route itself could serve as a source of such updates. This would involve saving the current filter state, rerunning the prediction step using points along the selected route as pseudo-GPS updates, and then continuing the correction. This step would occur after the route has been selected, but before applying the route to replace the predicted trajectory.

However, in practice this approach failed due to difficulties in assigning accurate timestamps to the route points. Proper timestamps are crucial because the Kalman filter processes measurements strictly in chronological order. For example, if a GPS point is one second ahead of the current state, the filter will predict one second forward and then incorporate the GPS position in its update. But if that same point is incorrectly given a timestamp two seconds into the future, the filter will instead predict two seconds ahead which leads to a predicted position that is further from the truth.

When the filter then receives the GPS update after two seconds of prediction, the difference between the predicted and measured positions appears much larger than it actually is. Since GPS measurements usually have lower uncertainty, the filter trusts and puts more weight into the GPS and applies a strong correction. But this correction is based on the false premise that the prediction wasn't inaccurate in terms of position, just mistimed. The result is a distorted estimate that pulls the filter away from the true state, rather than toward it.

Another problem caused by a point with an incorrect timestamp is how it effects subsequent points. The effect is that the subsequent GPS point lacks one second in its prediction since the previous update happened one second too late. Then the subsequent prediction will also be far off from its GPS point causing the filter to again do a hard correction. If the timestamp is incorrect enough it could even result in the positional updates being in the wrong order creating even worse corrections.

So even if using a route as positional updates has the potential to increase the accuracy of the prediction it should not be done until a more accurate way to interpolate timestamps for the route is found. One could also try to look into alternative ways to decide when to use the route points as positional updates in the Kalman filter. Alternative methods should work as long as they do not update with the route at inaccurate timestamps.

5.2.6.1 Distance based route interpolation

One theorised method for rerunning the prediction with the route is using predicted length instead of time as a metric. Since the prediction and route is of the same length one could use that fact to get an interpolation of where the route points should be in the series of input data. It would work by taking the distance to the next point in the route, predict until the predicted length matches the distance and then update the Kalman filter using the route point. If the first prediction has a good accuracy the route points should be applied accurately which lowers the accumulated error caused by the lack of positional updates.

5.2.7 Machine learning assisted accelerometer calibration

The accelerometer is not necessarily mounted on the vehicle in such a way that all the axes perfectly align with the car's axes. It is also possible for it to be mounted offset from the vehicles centre of mass causing it to possibly give wrong readings.

Because of this, the accelerometer needs to be calibrated. One way to do this would be to make use of machine learning. The machine learning algorithm could simply be fed data from the GPS, accelerometer and other sensors and use that to correctly apply the proper rotation matrix on the accelerometer.

5.2.8 3D tracking

Currently, the Kalman filter tracks the vehicles movement in only two dimensions, while the real-world driving pattern of the car will be in three dimensions. This means that the filter accounts for yaw but neglects pitch, even though both factors affect measurements of the car's acceleration and speed. This could lead to inaccuracies in the predictions.

Consider a scenario of a car driving at a speed of 50km/h with a 10° downward tilt due to a sloped road. The two dimensional Kalman filter will not take this tilt into consideration. This means that when receiving a measurement of 50km/h it assumes the car is moving forward at that speed when in reality the forward speed is

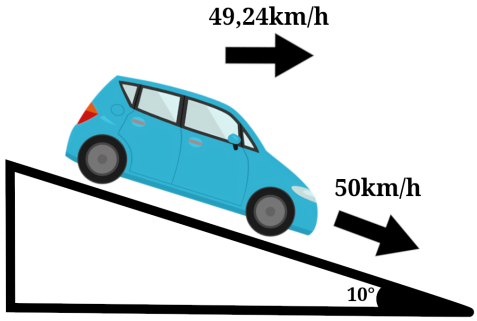

Figure 5.2: Illustration showing the difference between the cars speed relative to a slope.

49,24km/h. Consequently, the filter's perception of the car is off by 1,5%. This could be improved by modelling the cars movements in three dimensions and including the cars pitch in the state.

5.2.9 Parallel execution for route evaluation

The algorithms current backtracking approach has the potential to degrade performance. When the algorithm must backtrack through multiple previously evaluated segments, it may create noticeable processing delays that could impact system responsiveness.

One idea to solve this is implementing a parallel execution framework that simultaneously evaluates multiple high-probability routes rather than sequentially testing and potentially backtracking. At decision points where multiple routes have similar scores (such as highway exits or complex intersections), the algorithm would spawn parallel processes to follow each promising path.

An example could be an exit on the highway where both taking the exit and staying on the highway are scored similarly. This means that choosing either option could result in an incorrect prediction resulting in backtracking. But if instead of choosing one option the algorithm is run on both options, the need for backtracking the wrong option is not necessary since both have been computed in parallel. When one thread runs into a situation with no plausible routes, it will just end its process and assume the other option was the correct one. As long as there is enough computing power to run both options in parallel in real time this will eliminate the need for backtracking and delays created by it when running in real time.

5.2.10 Benchmarking on multiple cars and multiple drivers

The data for benchmarking has only been collected by one car and one driver. This could cause the benchmarking to not be representative of the real world since the driving pattern and data could differ between different cars and drivers. To get more representative results, data from multiple vehicles and drivers should be collected and benchmarked on.

5.2.11 Benchmarking in multiple locations

As with the data only being from one car and one driver it is also from a quite limited area. The highway and country data is mostly from between Gothenburg and Sälen while the city data is mostly from the Gothenburg/Mölndal area. The behaviour of the algorithm could differ depending on the road and city infrastructure as well with left hand driving. Therefore to get more representative results the data being benchmarked on should be from a large variety of locations.

5.2.12 Combination of different scoring methods

As shown in 5.1.4, DTW has certain disadvantages when used for scoring in the algorithm. The specific scenario brought up in that 5.1.4 would not exist if yaw scoring was used since a sharp 90 degree turn would yield a higher score than going straight ahead. But yaw scoring has its own set of disadvantages so using that instead causes more problem which was shown during development. One idea is to use a weighted combination of different scoring methods to get the best of both. Then a low score would only be achieved if the route has a low yaw score and low DTW score. This was not implemented due to the time it takes to figure out good weights for the combination.

5.2.13 Machine learning assisted parameter adjustment

In the algorithm there are many parameters that need to be adjusted for optimal results. Instead of doing manual benchmarking tests and selecting the optimal combination of variables from that, one could instead design a machine learning system that runs the algorithm and learns the best values for all of the parameters.

5.2.14 Dynamically setting parameters

In the algorithm there are many parameters that affect how the system performs. While some might be better as static constants, parameters such as the sliding window size or predictions size could gain from being set dynamically.

One example when this could be useful is in the case of the car driving on a straight road for an extended period. Here no alignment will be made since all alignments will be of a straight road meaning the algorithm prefers the suggested route without alignment. But if the prediction is consistently too short or too long the algorithm will drift more and more backwards or forward as the car continues to drive on a straight road. This means that when the car actually makes a turn the algorithm could be so far back or forward that the sliding window is too small to notice the actual turn the car is taking. This results in the algorithm missing the turn which could lead to incorrect predictions or a backtracking to the root node meaning no route could be found. If the algorithm assumes that its prediction is consistently too short or too long it will know that the longer it runs without choosing any alignment the more the algorithm has drifted forward or backwards. If it then uses this fact and increases the sliding window size each time it doesn't make an alignment adjustment

this backward or forward drift could be more likely noticed due to the larger sliding window size having a larger opportunity to detect misalignments.

5.2.15 Correction after gaining GPS

When the vehicles GPS is regained the algorithm will start estimating using the GPS signal. This will correct that error and highly accurate estimates will be made again. But this only applies to estimates after the GPS signal is regained and not afterwards.

An improvement to the algorithm could be to retroactively go back and correct previous estimates based on the regained GPS signal. For example if the regained GPS signal is on the same route but 100 meter ahead or behind the previous estimates would be extended or shortened to match up with the GPS.

Another more complex example is if the GPS shows up on another road than the estimate, indicating the algorithm has chosen the wrong route. A possible solution to this is forcing the algorithm to backtrack and chose another route and estimate until the point where the GPS is regained. If it on the same road as the GPS it is corrected but if it still is on the wrong road it will need to backtrack again and continue until it has found the correct route. If the algorithm choose the wrong route late in its prediction it should quickly find the correct route with this method. If the error happened early and the GPS outage was long this method could run for many iterations since the amount of possible routes could be very high.

5.2.16 Parking

Since our algorithm only focuses on vehicles driving on road networks it is not able to handle when a vehicle turns into a parking lot, parking garage or other areas outside of road networks. When entering an area outside the road network the algorithm will still try to predict the vehicles movement which could lead to errors. If the car moves inside of this area the algorithm's prediction might move away from that area instead of standing still giving a misleading position estimate as shown in 4.4

5.3 Conclusion

In this thesis a dead reckoning algorithm for vehicles driving on road networks, using a Kalman filter and a custom map correction algorithm, was presented. There are future works which could improve the algorithm or its implementation in different ways. But even with future works the proposed algorithm provides a high likelihood of correct position estimation for outages of upwards of one minute. This has been shown through the benchmarking of real world data by running the algorithm on multiple scenarios created from 185 different car data files. Through benchmarking it is also shown how the algorithm is better than purely using a Kalman filter. Which is how this thesis has shown that by utilising road networks through prediction, comparison and route selection, corrections can be made to the Kalman filter improving its accuracy in position estimation.

Bibliography

[1] W.-W. Kao, "Integration of gps and dead-reckoning navigation systems," in *Vehicle Navigation and Information Systems Conference, 1991*, IEEE, vol. 2, 1991, pp. 635–643.

[2] D. Yohan, P. Merriaux, P. Vasseur, and X. Savatier, "Vehicle positioning in road networks without gps," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, 2015, pp. 1803–1809.

[3] C.-W. Tan and S. Park, "Design of accelerometer-based inertial navigation systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 54, no. 6, pp. 2520–2530, 2005.

[4] E.-H. Shin, "Estimation techniques for low-cost inertial navigation," 2005.

[5] B. Barshan and H. F. Durrant-Whyte, "Inertial navigation systems for mobile robots," *IEEE transactions on robotics and automation*, vol. 11, no. 3, pp. 328–342, 1995.

[6] M. Brossard, A. Barrau, and S. Bonnabel, "Ai-imu dead-reckoning," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 4, pp. 585–595, 2020.

[7] N. Yu, X. Zhan, S. Zhao, Y. Wu, and R. Feng, "A precise dead reckoning algorithm based on bluetooth and multiple sensors," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 336–351, 2017.

[8] N. Dicu, G.-D. Andreescu, and E. HoratiuGurban, "Automotive dead-reckoning navigation system based on vehicle speed and yaw rate," in *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, IEEE, 2018, pp. 000 225–000 228.

[9] S. K. Gehrig and F. J. Stein, "Dead reckoning and cartography using stereo vision for an autonomous car," in *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)*, IEEE, vol. 3, 1999, pp. 1507–1512.

[10] J. Wahlström, I. Skog, J. G. Rodrigues, P. Händel, and A. Aguiar, "Map-aided dead-reckoning using only measurements of speed," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 3, pp. 244–253, 2016.

[11] X. Hou and J. Bergmann, "Pedestrian dead reckoning with wearable sensors: A systematic review," *IEEE Sensors Journal*, vol. 21, no. 1, pp. 143–152, 2020.

[12] M. E. El Najjar and P. Bonnifait, "A road-matching method for precise vehicle localization using belief theory and kalman filtering," *Autonomous Robots*, vol. 19, pp. 173–191, 2005.

[13] P. D. Groves, *Principles of GNSS, inertial, and multisensor integrated navigation systems*. Artech House, 2013.

[14] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[15] A. Zucconi, "The mathematics of the kalman filter," 2022, Accessed: 2025-04-10. [Online]. Available: `https://www.alanzucconi.com/2022/07/24/kalman-gain/`.

[16] P. Abbeel, A. Coates, M. Montemerlo, A. Y. Ng, S. Thrun, *et al.*, "Discriminative training of kalman filters.," in *Robotics: Science and systems*, vol. 2, 2005, p. 1.

[17] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," in *Handbook for automatic computation: volume II: linear algebra*, Springer, 1971, pp. 134–151.

A

Appendix 1: Quadrant analysis of benchmark results

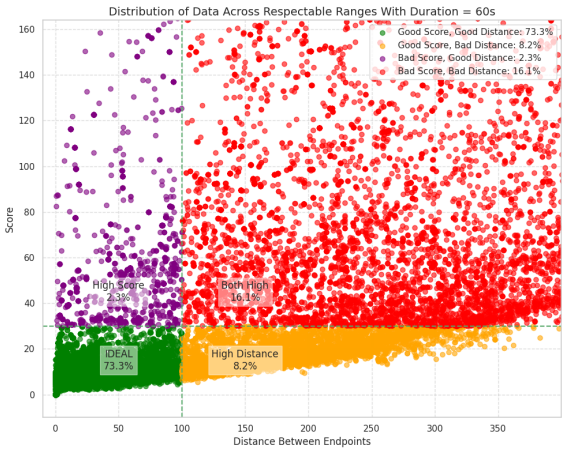

Figure A.1: Quadrant distribution for 60 second GPS outage

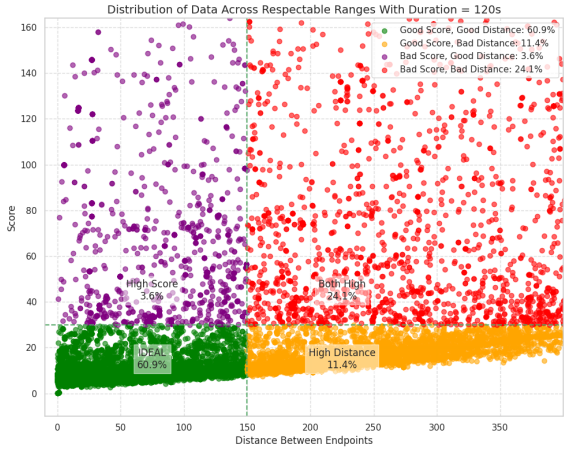

Figure A.2: Quadrant distribution for 120 second GPS outage

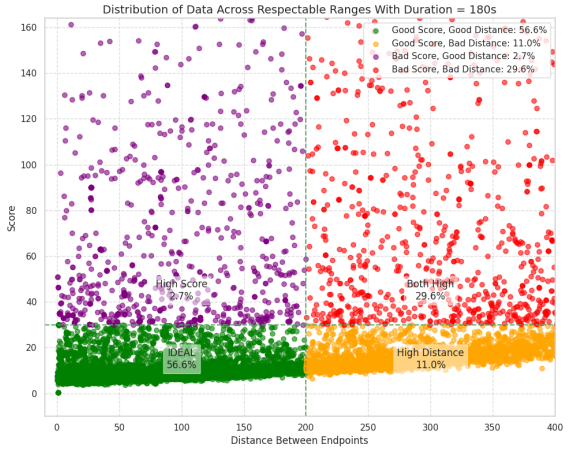

Figure A.3: Quadrant distribution for 180 second GPS outage

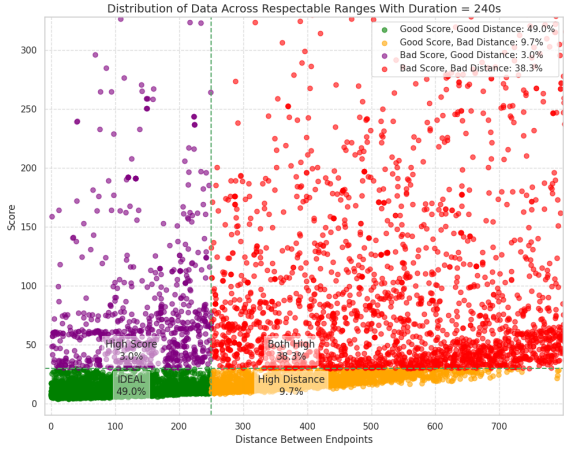

Figure A.4: Quadrant distribution for 240 second GPS outage

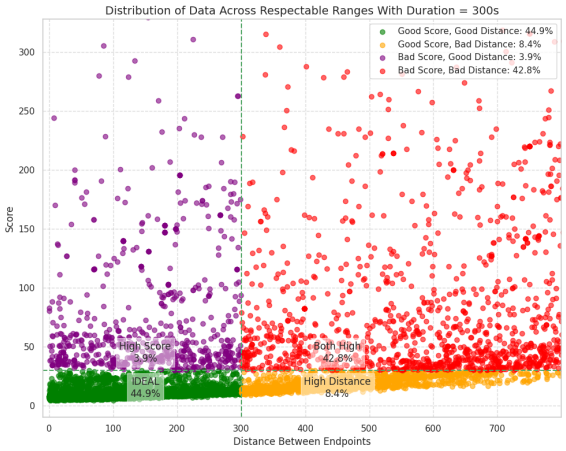

Figure A.5: Quadrant distribution for 300 second GPS outage

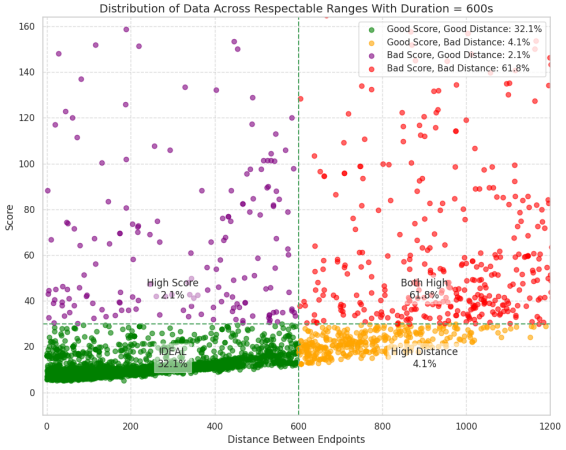

Figure A.6: Quadrant distribution for 600 second GPS outage

B

Appendix 2: Road type comparison

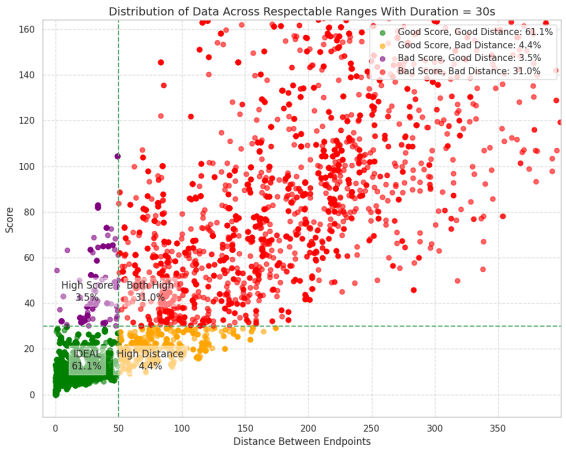

Figure B.1: Quadrant distribution for city roads.

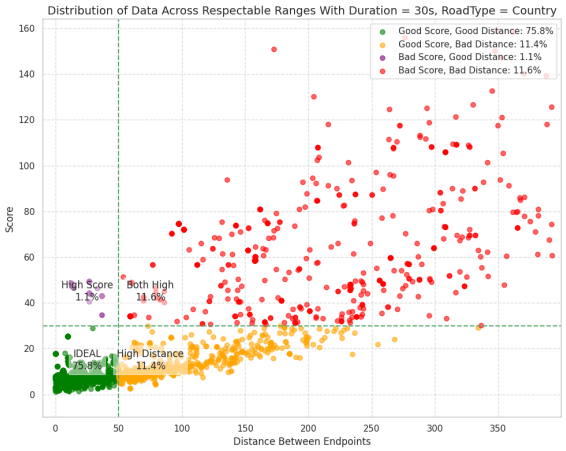

Figure B.2: Quadrant distribution for country roads.

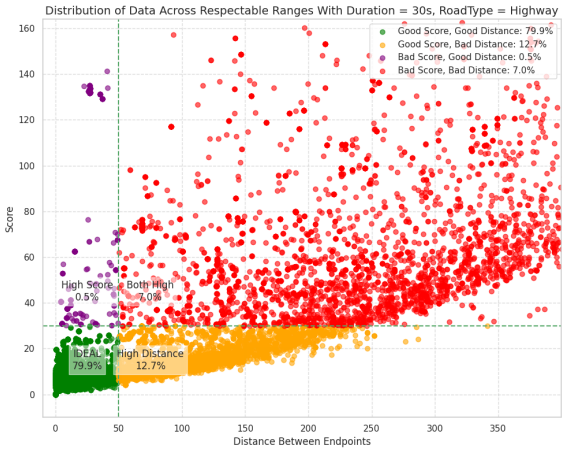

Figure B.3: Quadrant distribution for highway roads.

C

Appendix 3: Success rate by parameter configuration and road type

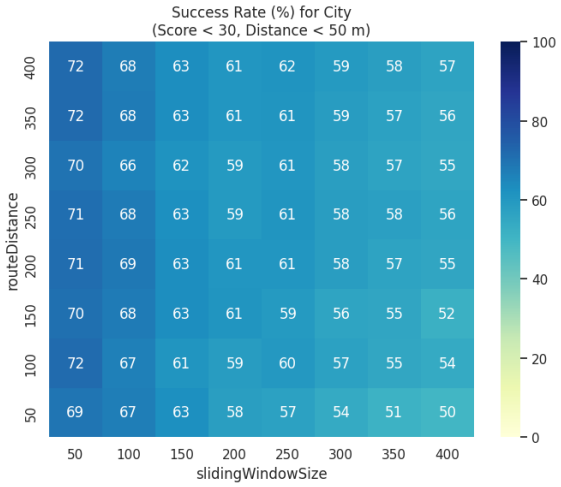

Figure C.1: Heatmap showing success rate for different parameter configurations when run on city roads.

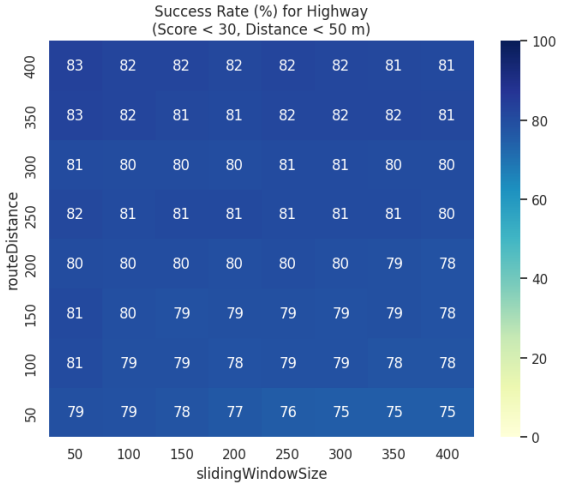

Figure C.2: Heatmap showing success rate for different parameter configurations when run on country roads.

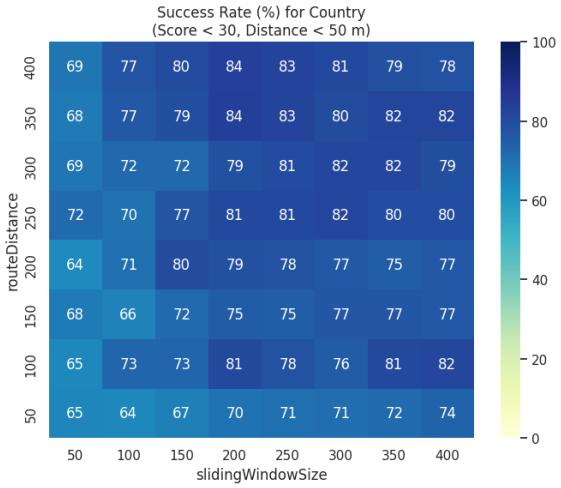

Figure C.3: Heatmap showing success rate for different parameter configurations when run on highway roads.