Offboard Camera Placement for Autonomous Robot Navigation

Joshua G. Send
Trinity Hall

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for Computer Science Tripos, Part III

University of Cambridge
Department of Computer Science and Technology
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: js2173@cam.ac.uk
Supervisor: Dr. Amanda Prorok

June 1, 2018

Declaration

I, Joshua G. Send of Trinity Hall, being a candidate for Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed:

Date:

Offboard Camera Placement for Autonomous Robot Navigation

Abstract

The conventional approach to autonomous vehicles is to use extremely perceptive robots that independently localise, analyse and navigate their environment. I consider an alternative: moving the localisation task to a network of off-board cameras installed throughout the environment. We can imagine that in urban areas in particular, such a network could be used by a range of autonomous vehicles including cars, bicycles and smaller delivery robots.

To examine feasibility, I analyse how performant cameras and vision algorithms need to be to adequately localise autonomous vehicles and keep them on a desired path. I build a model of camera-based localisation error, considering positional and orientational installation inaccuracy, as well as algorithmic imprecision and the camera's intrinsic properties. Using the error model, I show that a camera needs to be installed to within 10 centimeters and a 1 degree cone of its intended position and orientation, and that algorithmic error of about 10 pixels can be tolerated when locating a vehicle in the pixel plane.

From there, I move to optimise the placement of individual cameras. This was done by building a simulation of an autonomous vehicle following a path of piecewise constant curvature. The vehicle navigates the pre-planned path using an Extended Kalman Filter, and steering and speed controllers. A network of cameras is included in the simulation by introducing a ray tracing module, sending the vehicle location updates (along with uncertainty) whenever it is spotted by a camera. The update is used to correct the vehicle's own dead-reckoning estimate of its position, and in turn influences its navigation via the feedback controllers.

I examine four degrees of freedom for the placement of a single camera viewing a road: pitch, yaw, and horizontal and vertical resolution. I present a heuristic, inspired by Kalman gain, which is used to filter the large search space. The remaining configurations are evaluated with principled metrics which utilize the full simulation I developed. These metrics are then also used to examine the placement of multiple cameras along a predefined path. Results show that cameras should face along a road, rather than perpendicularly, and match the road to the diagonal of their field of view. I also discuss the costs and benefits of various optimisation metrics.

Word count: 11911

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Vision

It is conceivable that, in the near future, private vehicles would have been made redundant by an efficient fleet of local autonomous vehicles, ferrying passengers on demand between locations in a city. The details of the implementation could vary – the current, conventional approach is to use highly perceptive vehicles that completely sense their environment independently. I propose an alternative direction, where the city manages a public network of cameras, helping vehicles localize themselves in the streets and sense their environment.

To imagine further what this might look like, a city could offer a pool of robots consisting of vehicles from various manufacturers, managed by a public-private partnership. Much like a utility, this service comes to be seen as an essential part of the city infrastructure fulfilling the needs of its citizens.

By the time this vision could be implemented, object detection algorithms would be highly accurate, enabling cameras to identify pedestrians and vehicles to within a few centimetres. These locations would be transmitted to nearby vehicles. As a result, GPS, radar, and expensive LIDAR systems would be largely redundant on the autonomous vehicles.

Following from this, autonomy becomes cheaper to implement, and more accessible. Vehicle manufacturers work with the utility to keep the cameras functional and up to date. City GPS canyons are avoided, and autonomy is enabled on everything from bicycles, to small delivery robots, to standard vehicles – all by shifting the sensing task from the individual user to the infrastructure.

1.2 Motivation

The core task of an autonomous vehicle is to navigate correctly and safely from a start to a destination. This process can be split into major subtasks consisting of path planning, localization, and safe navigation.

Current autonomous vehicle projects use a large set of sensors, combined with maps, to solve each of those complex tasks. This comes at a cost: current Level 3 attempts, for example, use LIDAR, which provides rich data to perform self-localization and dynamic obstacle avoidance, but also costs around $75,000 [38]. The computational power required to fuse and process data from wheel odometry, GPS, inertial measurement units (IMUs), cameras, radar, LIDAR is also substantial and generally implies only large platforms can be used.

I propose moving the localization and navigation sensing tasks from vehicles to the infrastructure, which is particularly applicable in urban environments. I examine a static environment containing a single vehicle, with dynamic components and safety to be pursued in future work.

Most current localization approaches utilize global satellite navigations systems, such as GPS and its augmentations like real-time kinematic positioning (RTK) [6]. The US government reports a consumer-grade GPS accuracy of 0.715 meters 95% of the time [32]. However, GPS errors are introduced in urban environments – according to Miura et al., even with the authors' proposed correction steps, the mean localization error was around 5.2 meters in urban canyons, where multipath effects or buildings block signals from

multiple satellites. In contrast, most authors agree autonomous vehicles can tolerate **0.2 to 0.35 meters** lateral error [37][30][19].

To achieve these error targets, current techniques may use computationally heavy visual odometry [30], very high precision maps [19], or expensive LIDAR to localize lane markings very accurately [29].

In this work, I show that if certain installation and algorithmic constraints can be met, offboard cameras mounted at the side of roads can provide the required localization accuracy without reliance on GPS, LIDAR, or radar. I also build a simulation of an autonomous vehicle which utilizes wheel odometry-based dead reckoning to navigate between updates received from the infrastructure. In this context, I then examine the task of optimizing offboard camera placements and configurations for localization and navigational performance.

1.3 Contributions

The practical questions answered by this work are the following:

1. How accurately do offboard cameras need to be installed to provide good vehicle localization? How accurate do the computer vision algorithms recognizing vehicles need to be to provide good vehicle localization?

2. In which direction should an offboard camera be facing and what should its properties be (for instance, field of view) to optimally aid vehicle localization and navigation, and how are they related to road curvature?

3. Given a road or path to follow, how should a set of cameras be placed along the path to optimize navigational performance?

The first item relates to the feasibility of offboard camera-based localization and is addressed at the end of Chapter 4.

The second and third points examine the optimal camera placement task

for localization and navigation performance. This is related to past work on landmark placement for robotic navigation, where the robots observe landmarks fixed to the environment. Here, the use case is inverted to observe the robot as it moves through the environment instead. Thus, it is also closely related to previous work on surveillance networks.

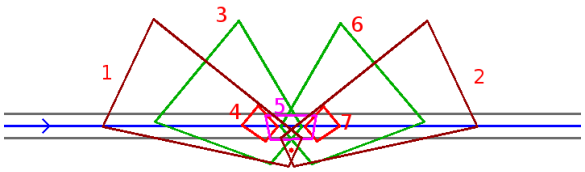


Figure 1.1: Which camera configuration (each represented by a colourful trapezoid) is best for a vehicle navigating along the blue line?

While literature on surveillance using cameras has explored camera placement, few have allowed more degrees of freedom than simply adjusting the tilt (pitch) and choosing one of a limited set of positions. Surveillance tasks normally seek to maximise objective functions other than robotic navigational performance, which is the goal here.

The overall novelty of this work is as follows: analysing installation and algorithmic requirements for offboard cameras to localize vehicles, and exploring more degrees of freedom in the sensor placement task than prior work, including examining the effect of the environment shape on camera placement. During the development of these topics, I make the following specific contributions:

- An error model for vehicle localization using offboard, infrastructure-mounted cameras, which is specifically motived by physical properties such as installation uncertainties and algorithmic error
- Using the developed error model, a feasibility analysis for using cameras for localization
- An analysis of placement of a single camera along roads, and the relationship to road curvature

- An analysis of placement of sets of cameras along roads

1.4 Overview

Chapter 2 reviews relevant literature in the fields of localization, landmark placement, and surveillance. Chapter 3 presents the simulation used for analysis, as well as background material referenced throughout this dissertation.

Chapter 4 then develops the error model for cameras used in simulations, and examines the feasibility of using cameras in the infrastructure for localization. Chapter 5 optimizes single and multiple camera placements in various environments and using different objective functions. This is followed by concluding remarks.

Chapter 2

Related Work

This work touches on a wide variety of fields, from control theory to computer vision to landmark selection. I focus on selected works which motivate the problems being tackled here.

2.1 Offboard Localization

A core idea underlying this dissertation is the use of external sensors to correct the localization of a mobile robot which otherwise primarily uses dead reckoning to navigate. This is hardly a new idea – sources since the 80s and 90s have looked at using infrastructure-based sensors for localization, navigation, and tracking.

The choice of sensor, however, varies widely. Many researchers have used bearing or distance-based triangulation methods combined with ultra-wide band (UWB), infrared, or even sonar to perform localization [21]. These all suffer from the same drawbacks: measurements can be unreliable due to delayed signals or multipath effects, and interfacing with these systems requires specialized hardware. Received-signal-strength techniques, may also use triangulation, and suffer from similar problems and degrade further in dynamic environments [18].

Some passive approaches have been attempted as well, such as providing the vehicle or the environment with RFID tags that can be scanned and matched to a premade map, sometimes further combined with signal strength techniques [16]. According to Dr Ioannis Brilakis [personal communications, March 2018] at the Cambridge Department of Engineering, these approaches have generally been deemed a practical failure due to maintenance issues.

External, vision-based monitoring systems found some support in the 90s and early 2000s. The best-known work might be from Kruse and Wahl [4], who implement a network of cameras in an indoor environment to track and localize robots. They utilize a set of LED's mounted on the corners of the robot to aid in this, and show that they can reduce the robot's positional error. Later, Menegatti et al.[9] use a slightly different approach to matching images taken by the infrastructural cameras to those taken by the robot to perform localization. However, since that 2005 paper external camera-based localization appears to have fallen out of favour, only recently appearing slightly differently in the context of surveillance tasks.

The lack of further development of offboard camera localization is likely to be caused by two main difficulties: firstly, the cost of instrumenting the environment – but note that many cities already have traffic or security cameras installed throughout. Secondly, a major problem in the past was the inability to acquire the agent without visual markers on the robot. From [9], "... visual features are not stable and it is not easy to correctly detect the robots...". I believe with the recent breakthroughs in machine learning-based object recognition and segmentation, it is time to revisit the option of external cameras for localization.

There are added benefits to using cameras as well: they are more immune to crosstalk and interference than radio-based techniques, and provide a large amount of salient information. This can include information about other objects in the scene, such as pedestrians, and information beyond just (x, y) position, such as vehicle orientation.

2.2 Camera and Landmark Placement

A second theme is the optimization of sensor placement. This problem is addressed in surveillance literature, computational geometry, and robotics, among other fields.

Perhaps the most direct predecessor to this work is the PhD thesis of Beinhofer[28], which focuses on landmark placement for mobile robots. The key difference to my work is that he uses the conventional approach of a robot-mounted camera. Additionally, he primarily considers cameras with circular fields of view, looking straight up, whereas I consider position, orientation, and field of view as variables. However, I utilize many of his formulations, including the Monte Carlo approach to evaluating mutual information (Section 5.2.1). A noteworthy idea of his thesis is the use of submodularity (provably diminishing returns on each additional landmark) to guarantee good approximations to the NP-hard landmark placement problem using greedy optimization.

Another relevant paper from the surveillance literature is [15]. Although the authors formulate a completely different objective function intended to maximise the observed length of piecewise-linear paths, their ideas inspired my heuristic function presented in Section 5.1. In this work, I use my heuristic to search four degrees of freedom (pitch, yaw, and vertical and horizontal field of view), retaining parameters of interest. These are then passed onto a more complex metric based on Beinhofer's thesis.

In [12], the authors optimize infrastructure-based camera coverage of a map, modelling coverage as a 2D triangular region. They propose using binary mixed integer programming to find the optimum of multiple camera orientations. The work in [20] defines a general sensor network coverage problem, which optimizes generic directional sensors' field of view and yaw using linear programming, but specifically consider ground-based sensors rather than cameras. The reader is referred to [23] for more work on camera-based coverage models. Common to these works is their 2D treatment of the placement

problem, optimizing for coverage rather than localization capability, as well as using expensive linear programming-like optimizations.

Lastly, the work from Bansal, Badino, and Huber [27], considers the problem of orientation for cameras mounted on autonomous vehicles. They show that the most informative orientation (defined by entropy) is given when the camera maximises useful pixels – i.e., focuses the most pixels on informative regions that have distinguishing characteristics, mostly along the edge of roads.

Chapter 3

Design and Implementation

This section outlines the work completed during the development of this dissertation, and provides mathematical background referenced throughout the report. Code can be found at `https://github.com/flyingsilverfin/AV-Camera-Placement`.

3.1 Approach

The basis of this work is a simulation developed using industry-standard tools. The Gazebo [7] robotics simulator is used in conjunction with ROS, the Robot Operating System [17], to model a non-holonomic Ackermann drive vehicle following predefined paths. A camera is modelled using standard formulations discussed in Section 3.2.

Figure 3.1: Render of the Ackermann drive model used in simulation.

An expensive, high fidelity simulation was used instead of a real hardware implementation. Simulation allows quickly modifying the environment and models to examine a greater range of tasks.

3.2 Cameras

I implement a standard pinhole projective camera model as a raytracing module. Raytracing allows adding parametric geometry into the world, which is used to define road boundaries ("buildings"). Additionally, my formulation can also be used to model non-pinhole camera models, including ones that better represent ultra-wide field of view cameras. However, this work leaves examining the impact of such cameras on robot localization and navigation as a future research direction; I cap the maximum field of view of the pinhole model at 100 degrees horizontally and vertically, when the perspective model loses its realism [3].

In the presented scenario, a camera's task is to localize vehicles in world coordinates, and provide an error or confidence in each localization. To localize the vehicle, the camera needs to be provided with its own world location and orientation, as well as its parameters. *Camera parameters* will be used to mean both the horizontal and vertical field of view of the camera, as well as its resolution.

By analysing the image the camera sees, vehicles can be identified and a pixel estimate of the vehicle centre obtained. Knowledge of the camera's position and orientation is combined with the estimate, to calculate the centre of the vehicle on the ground plane. This estimate is transmitted to the vehicle to update its belief. I assume a planar surface at $Z = 0$, so updates from the camera are world locations (x, y).

In the following chapter I model an abstract computer vision algorithm for detecting vehicles. A concrete implementation could use an object detection or segmentation network, such as YOLO [39] or DeepLab [34] to determine a bounding box for the vehicle. The centre of the box in the pixel plane can be seen as an estimate of the volumetric centre of the vehicle. Using an estimate of the height of the vehicle, a geometric correction can be applied to obtain a ground estimate of the vehicle position.

However, real implementations could be arbitrarily complex algorithms, in-

cluding machine learning algorithms specifically trained for this task.

3.2.1 Coordinate Systems

For reference, Figure 3.2 shows the orientations associated with camera placements. I only allow pitch and yaw to be modified during my optimizations (in green), as it simplifies the geometric modelling performed in the following chapters.

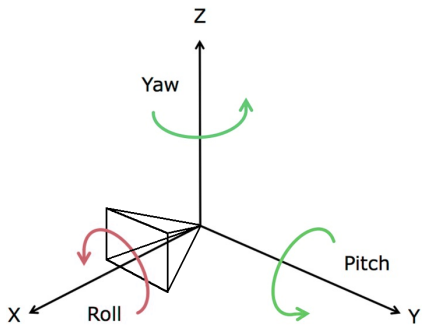


Figure 3.2: Camera orientations referred to as roll, pitch, and yaw. Roll is not modified in this work, only pitch (tilt downwards) and yaw (rotation about Z). The camera symbolized here is conceptually at 0 degrees roll, pitch and yaw

3.3 Road Representations

One key aspect of this work is that the desired trajectory is known. There are a variety of ways of representing paths that have been used in the literature.

One simple option is to discretize any curve as a set of linear piecewise segments such as in [15], where it was used to represent paths to observe in a surveillance task. However, it require trading off accuracy and number of segments. I chose to use a slightly more complex representation: piecewise constant curvature paths, also known as Dubins curves [1]. They have an easy geometrical definition, where curvature $C = 1/R$, R being the radius of the circle with the given curvature.

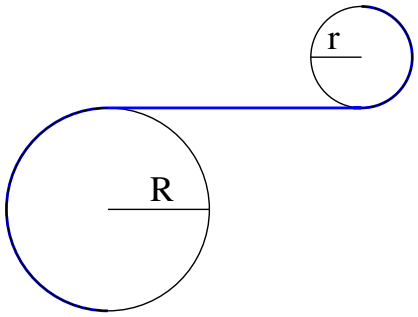


Figure 3.3: A Dubins curve with a segment with small curvature, large radius (R), a straight line segment with 0 curvature ($R = \infty$), and a large curvature, small radius (r).

Dubins curves can therefore be easily parametrized using only curvature and length. My implementation joins together sequences of constant curvature segments to form continuous curves. Dubins curves lend themselves to analytically finding the closest point on the curve to any other world point. I used this property to calculate the perpendicular error distance used for the steering controller discussed in 3.4.

Alternative representations were considered, including splines and Euler curves (linearly varying curvature paths), but these were deemed overly complex for the purpose of this work.

3.3.1 Implementation

In order to provide road boundaries I define constant-width roads (3 meters), with an additional side offset that might represent pavements (1.5 meters). Cameras can be placed at the side offset. I then place parametric cylindrical walls or planes in the ray-tracing space to prevent cameras from seeing beyond the edges of roads.

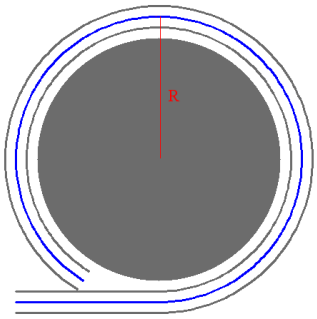


Figure 3.4: A Dubins curve with road boundaries at $\pm 1.5m$, and a cylindrical "building" block which obstructs the ray tracer from seeing the road on the opposite side of the circle.

3.4 Vehicle Model and Controls

The navigation tasks performed throughout are executed by a modified open-source model of a Toyota Prius [35]. This model takes two primary commands: throttle, and steering wheel angle. The state of the robot is both a world position (x, y, z) and orientation quaternion. Due to the planar restriction though, the state we are interested in can be written as (x, y, θ), where heading θ is the angle from the $x = 0$ axis on the ground plane.

3.4.1 Controllers

PID Controller

PID controllers [2] are widely used feedback controllers that calculate a response based on an error signal, its integral, and its derivative (hence, **P**roportional, **I**ntegral, **D**erivative).

To use a PID controller, one needs to define target value and a current state, which together form an error. Then, the PID gains are tuned to match the response of the system.

Speed Controller

The speed controller is a PID instance with access to the throttle and current vehicle speed. Thus, I need to define a target speed for all points in time to create an error function.

I set target speed policy s based on the curvature C of the closest point on the reference path:

$$s(C) = \begin{cases} 20.0 & C = 0.0 \\ \frac{k}{C} & C > 0.0 \end{cases} \tag{3.1}$$

Constant k was experimentally found to perform well at $k = 0.5$. 20.0 m/s was also found to be a reasonable straight-line speed, which corresponds to about 72 km/h.

However, just using $s(C)$ as the target speed results in large overshoots at tight corners – velocity controllers practically require some amount of lookahead due to finite decelerations. Therefore, I define the target speed as the mean of $s(C)$ at n equidistant points ahead of the vehicle on the curve. This induces a linear deceleration up to a curve. I only allow accelerations once the curvature decreases.

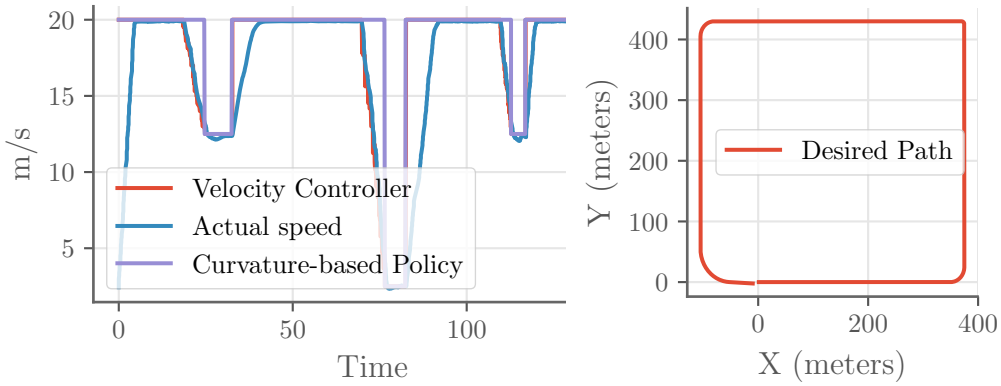


Figure 3.5: Speed controller performance. Notice the sharp corner (5 meter radius) in the top right corner corresponds to a big drop in speed.

Steering Controller

The steering controller must minimize deviation from the desired trajectory as much as possible. PID controllers are difficult to use for this task since they tend to oscillate or underreact, or perform well at one constant speed but not others. Instead, I implement the steering controller published in [14],

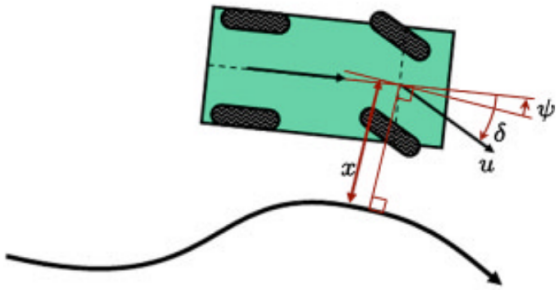


Figure 3.6: The original figure from [14] which describes the crosstrack or lateral distance (here x), and Ψ the angle between the current orientation and the tangent to the curve.

which was used by the autonomous vehicle that won the 2006 DARPA Grand Challenge. They define a controller that directly computes the desired steering angle using only the perpendicular distance to the path, as well as the vehicle's speed $u(t)$.

$$\delta(t) = \Psi(t) + \arctan \frac{ke(t)}{u(t)} \tag{3.2}$$

The original paper shows that when used with a linear bicycle model, the crosstrack error $e(t)$ converges exponentially to zero. The constant k was experimentally determined to perform well at $k = 0.95$ across a wide variety of paths. Some overshoot can be found when entering mid-curvature bends, which is a consequence of using two disjoint controllers for steering and velocity. A more sophisticated model might use model predictive control for all required controls [8].

3.4.2 Extended Kalman Filter

The simulated vehicle tracks its position using an Extended Kalman Filter (EKF) to integrate camera updates and correct its inaccurate dead reckoning [26]. An EKF operates in predict and update steps, propagating the vehicle's state according to a motion model in the predict step, and correcting the vehicle state during the update step if there is a measurement present. Like a standard Kalman Filter, every step produces three dimensional normal distributions over the mean belief of the robot (x, y, θ) The main difference to a standard Kalman Filter is that non-linear motion models can be used by linearising the model at each timestep with a Taylor-like expansion about the estimated mean.

Formally, if the previous mean belief (i.e., its state) of the vehicle is $\boldsymbol{x}_{t-1} = (x_{t-1}, y_{t-1}, \theta_{t-1})$ and covariance is $\boldsymbol{\Sigma}_{t-1}$, the predict step is given as

$$\hat{\boldsymbol{x}}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) \tag{3.3}$$

$$\hat{\boldsymbol{\Sigma}}_t = \boldsymbol{F}_t \boldsymbol{\Sigma}_{t-1} \boldsymbol{F}_k^T + \boldsymbol{Q}_k \tag{3.4}$$

Here, f is the vehicle motion model described in the following section, $\boldsymbol{u}$ are control commands, and $\boldsymbol{Q}$ represents process noise. $\boldsymbol{F}$ is the Jacobian of the motion model, evaluated at the prior timestep – this is the linearisation step of the EKF that distinguishes it from a normal Kalman Filter.

The update step is then performed as the following, given a measurement $\boldsymbol{z}_t$.

$$\tilde{\boldsymbol{y}}_t = \boldsymbol{z}_t - h(\hat{\boldsymbol{x}}_t) \tag{3.5}$$

$$\boldsymbol{S}_t = \boldsymbol{H}_t \hat{\boldsymbol{\Sigma}}_t \boldsymbol{H}_t^T + \boldsymbol{R}_t \tag{3.6}$$

$$\boldsymbol{K}_t = \hat{\boldsymbol{\Sigma}}_t \boldsymbol{H}_t^T \boldsymbol{S}_k^{-1} \tag{3.7}$$

$$\boldsymbol{x}_t = \hat{\boldsymbol{x}}_t + \boldsymbol{K}_t \tilde{\boldsymbol{y}}_t \tag{3.8}$$

$$\boldsymbol{\Sigma}_t = (\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{H}_t) \hat{\boldsymbol{\Sigma}}_t \tag{3.9}$$

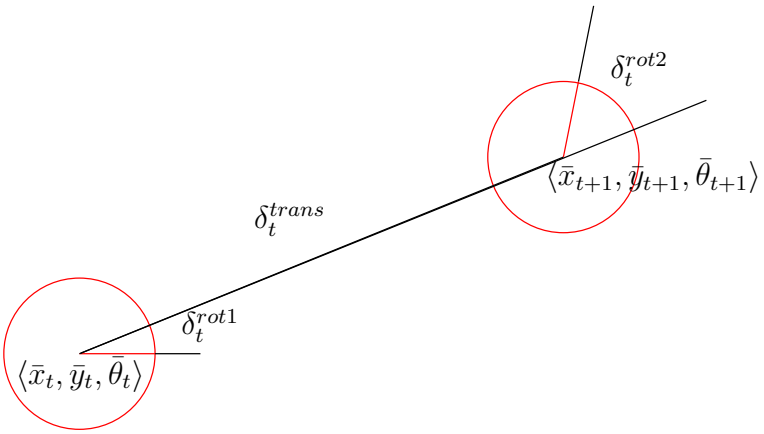
Here, $h(\boldsymbol{x})$ is the *sensor model*, which describes the relationship between a state and a produced measurement. $\boldsymbol{H}$ is the Jacobian of $h(\boldsymbol{x})$, which for my purposes I set as the identity (i.e. a linear relationship, which works well enough in practice). Finally, $\boldsymbol{R}$ is the covariance of the measurement.

An interesting value is the *Kalman Gain*, $\boldsymbol{K}$. In an intuitive sense, the update step can be seen as performing a weighted average between the vehicle's state, and the incoming measurement. The weighting $\boldsymbol{K}$ is determined by a relationship analogous to $\frac{\boldsymbol{\Sigma}}{\boldsymbol{\Sigma}+\boldsymbol{R}}$, if these matrices were scalars. I come back to this idea in Chapter 5.

Odometry Motion Model

I describe the probabilistic motion model $f(\boldsymbol{x}, \boldsymbol{u})$ that is used as the basis of the EKF predict step [10]. The control commands $\boldsymbol{u}$ are not actually the commands sent to the vehicle, but taken from the "wheel odometry", which is just the simulation's ground truth state and influenced by the vehicle controller (a common formulation). So, let $\boldsymbol{u}_t = (\bar{x}_t, \bar{y}_t, \bar{\theta}_t)$ be the vehicle's true state at time t.

The high-level explanation of the motion model is as follows: I use the vehicle's ground truth movement between timesteps to calculate a series of deltas, namely the translational change, and a starting and ending angular change. These are perturbed slightly and used to update the EKF's current state.



$$\delta_t^{trans} = \sqrt{(\bar{x}_{t+1} - \bar{x}_t)^2 + (\bar{y}_{t+1} - \bar{y}_t)^2} \tag{3.10}$$

$$\delta_t^{rot1} = \text{arctan2}(\bar{y}_{t+1} - \bar{y}_t, \bar{x}_{t+1} - \bar{x}_t) - \bar{\theta}_{t+1} \tag{3.11}$$

$$\delta_t^{rot2} = \bar{\theta}_{t+1} - \bar{\theta}_t - \delta_t^{rot1} \tag{3.12}$$

This calculates the ground truth changes to orientations and distance.

Next, noise is added to these deltas using a function $norm(\mu, \sigma^2)$ which samples the indicated normal distribution.

$$\hat{\delta}_t^{rot1} = \delta_t^{rot1} + norm(0, \alpha_1|\delta_t^{rot1}| + \alpha_2|\delta_t^{trans}|) \tag{3.13}$$

$$\hat{\delta}_t^{trans} = \delta_t^{trans} + norm(0, \alpha_3|\delta_t^{trans}| + \alpha_4|\delta_t^{rot1} + \delta_t^{rot2}| \tag{3.14}$$

$$\hat{\delta}_t^{rot1} = \delta_t^{rot2} + norm(0, \alpha_1|\delta_t^{rot2}| + \alpha_2|\delta_t^{trans}|) \tag{3.15}$$

Lastly, given current mean belief state (x_t, y_t, θ_t), generate the next EKF mean state as

$$x_{t+1} = x_t + \hat{\delta}_t^{trans} \cos\left(\theta_t + \hat{\delta}_t^{rot1}\right) \tag{3.16}$$

$$y_{t+1} = y_t + \hat{\delta}_t^{trans} \sin\left(\theta_t + \hat{\delta}_t^{rot1}\right) \tag{3.17}$$

$$\theta_{t+1} = \theta_t + \hat{\delta}_t^{rot1} + \hat{\delta}_t^{rot2} \tag{3.18}$$

This creates the updated state $\boldsymbol{x}_{t+1}$. The Jacobian $\boldsymbol{F}$ required for the EKF prediction step is computed from the partial derivatives of these three equations.

3.4.3 Calibration

There are four constants, α_{1-4} which are present in the odometry model equations. These determine the "shape" of the robot's probabilistic motion.

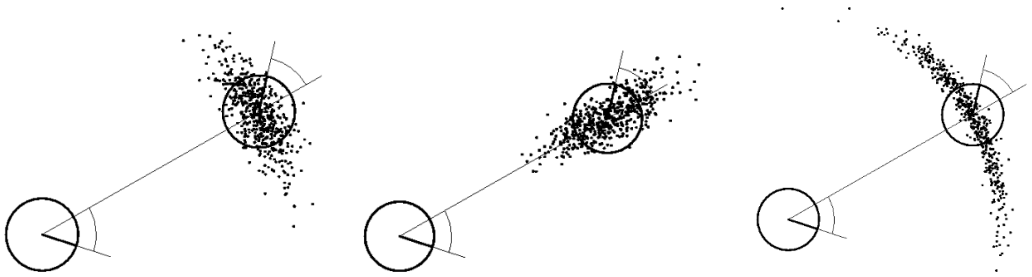


Figure 3.7: How various settings of α's affect the motion model. The canonical settings produce the leftmost spread[1].

[1] From http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/06-motion-models.pdf

I use settings which produce canonical distributions, increasing lateral uncertainty more than longitudinal uncertainty.

Since I am using a simulation without a physical reference vehicle to calibrate my parameters against, I use data from [37]. They measure their vehicle's wheel odometry over 12 laps of a circuit, 1800 meters long, similar to the one in Figure 3.8. I mainly aim to produce similar order-of-magnitude errors.

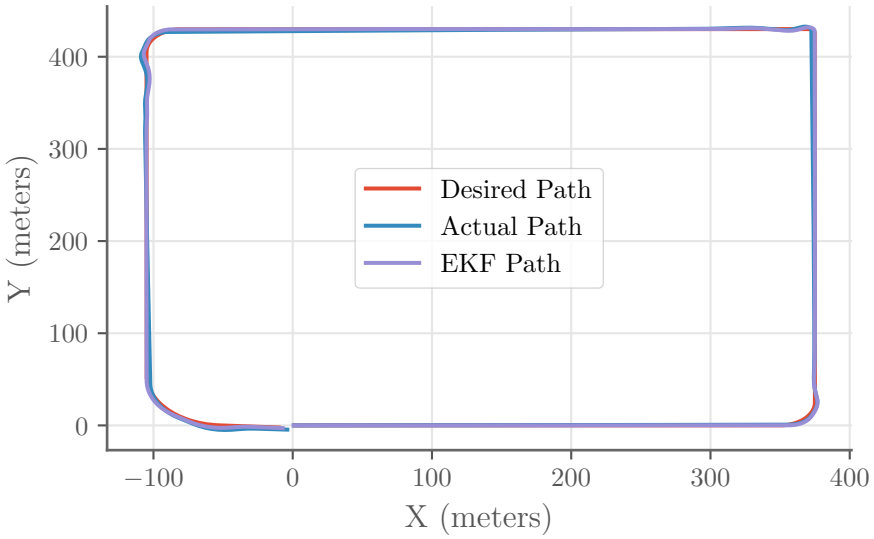


Figure 3.8: The calibration circuit containing one tight turn, two middle turns, and one low curvature turn, for a total length of about 1800 meters.

I ran my simulation on the circuit 20 times, and compare my calibrated

results to [37] in Table 3.1. The values are roughly aligned, so I use the corresponding values for α going forward.

Table 3.1: Mine versus Vivacqua, Vassallo, and Martins dead reckoning errors.

Metric	My Value	Value from [37]
Mean Final Positional Error	2.06m	2.4m
Max Final Positional Error	5.85m	5.3m
Angular Error Accumulation Rate	0.25 °/min	∼1°/min

3.5 Objective Functions and Optimization

This work approaches camera placement as an optimization problem, so the objective function needs to be defined clearly.

Past work has used a wide range of metrics for landmark placement, ranging from maximally reducing the trace of the robot's covariance matrix (an upper bound on location uncertainty) [25], to entropy of the final robot state [24], to the mutual information between robot states and observations [22]. Surveillance literature has also maximised the observability of a path, essentially defined as previously unseen path length in the camera's pixel plane [15].

Most authors discretise the space of possible sensor placements, since analytic or even differential solutions amenable to gradient descent are rare. The discretised problem is therefore categorized as an instance of the *constrained set size* problem, wherein the n best elements from a set of possibilities must be selected to maximise an objective function. Naturally, we are faced with exponential growth in the number of ways of placing n sensors: if the set of possibilities is V, the number of subsets of V with cardinality n is $\binom{|V|}{n}$, which is bounded below by $(\frac{|V|}{n})^n$. In fact, most variants of the problem are shown to be NP-hard [28].

The authors in [22] deal with the NP-hardness using *submodularity*, which is a diminishing returns property. If the objective function can be proved

submodular, the greedy algorithm is guaranteed to find a value within 63% of the global optimum. Beinhofer et al. show that their formulation of mutual information is submodular – I use this measure as my objective function combined with greedy optimization as well. I comment on submodularity in Section 5.5. I also examine using the trace of the covariance matrix as an objective.

Evaluating the mutual information even once is expensive, which is why authors have generally only optimized one or two degrees of freedom at a time. Since I aim to consider two orientational degrees of freedom, as well as field of view of the cameras, I require a means of cutting down the search space. In Section 5.1 I present a new, cheap heuristic which captures the localization capability of a single camera placement. These results influence the discretised space of orientations for the expensive greedy optimization.

3.5.1 Entropy and Mutual Information

To provide mathematical background for the mutual information measure, we need to understand entropy. Entropy can be seen as a measure of uncertainty of a random variable. The entropy of a discrete random variable $h(X)$ is maximised when every outcome is equally probably (a "flat" distribution). It is minimized when a specific outcome contains all of the probability.

In mathematical terms, entropy of a discrete random variable X taking on values $x_1, x_2..., x_n$ can be written as:

$$h(X) = -\sum_{i}^{n} p(X = x_i) log(p(X = x_i)) \tag{3.19}$$

Differential entropy extends the standard definition to continuous probability distributions, and may be negative if the probability density is greater than

one.

$$h(X) = -\int p(X = x) log(p(X = x))dx \tag{3.20}$$

Joint entropy of a set of continuous random variables $X_1, X_2...X_n$ is defined as

$$h(X_1, ...X_n) = \\ -\int p(X_1 = x_1, ...X_n = x_n) logp(X_1 = x_1, ...X_n = x_n)d(x_1, ...x_n) \tag{3.21}$$

Conditional entropy of random variable X, given a random variable Y is calculated as

$$h(X|Y) = h(X, Y) - h(X) \tag{3.22}$$

$$= -\int\int p(X = x|Y = y) log(p(X = x|Y = y))dxp(Y = y)dy \tag{3.23}$$

$$= \int h(X|Y = y)p(Y = y)dy \tag{3.24}$$

Finally, the mutual information between two random variables $I(X;Y)$ is defined as

$$I(X;Y) = h(X) - h(X|Y) \tag{3.25}$$

$$= h(Y) - h(Y|X) \tag{3.26}$$

Notice that this can be interpreted as the reduction in uncertainty of X, once Y is known. Or in other words, it is a measure of how informative Y is about X.

I will refer to these equations in Section 5.2.1.

3.6 Bringing it Together

The components described here are joined via ROS. Figure 3.9 shows how the components are utilized during the optimizations in Section 5.2.1. The dashed box including the cameras can be used standalone, as is done in Section 5.1. Everything except the Gazebo vehicle model was developed from scratch (modules in red). The green boxes represent inputs and outputs of the system.

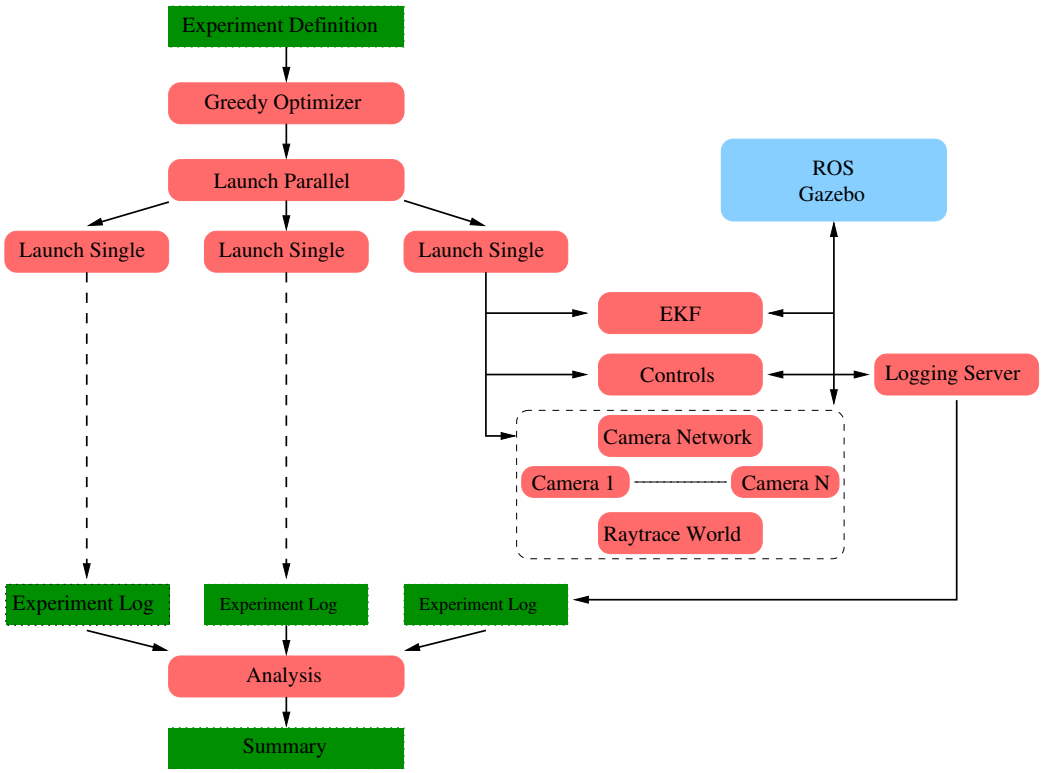


Figure 3.9: The full evaluation pipeline for computing simulation-based metrics.

The code is mainly Python, and so is not as efficient as it could be. Combined with the computational cost of Gazebo itself, I eventually decided to run all computations on Amazon AWS. To take full advantage of this I modified the system to be parallelisable, running many instances concurrently, collecting logs from each, and analysing them to extract whichever metrics are defined.

Chapter 4

Camera Model

This chapter develops the simulated camera which is used to report vehicle location and uncertainty. After validating the model, I use it to analyse absolute feasibility of off-board localization with cameras, and illustrate installation and algorithmic accuracy requirements. These steps do not require the full simulation, and only the ray tracing world, road definitions, and camera components.

4.1 Localization and Error

4.1.1 Approach

In a real-world implementation of the proposed system, a camera would wirelessly transmit estimated vehicle positions and associated uncertainty to vehicles in its field of view.

To enable position estimation, the camera needs to know its own world position and orientation, as well as the position and orientation of the ground plane. In this work, I assume the ground plane to be flat (a locally reasonable assumption, especially along roads), and equal to the $Z = 0$ plane. I report the position (x, y) of the vehicle as the estimated geometric centre of

the vehicle on the ground plane. Note that in practice, using the midpoint of the front or rear axle, rather than the centre, may be a more consistent reference between vehicles.

Once the computer vision algorithm running on the camera estimates the centre of the vehicle, we must derive an uncertainty in the form of a covariance. To start with, I develop a circular error radius, r, which I construct to contain the true error 95% of the time, but actually is about 99% accurate. While circular error radii are more amenable to analysis and are good estimates at steeper pitch angles, error ellipses are more realistic at low pitch angles and are an even more accurate error bound. Therefore, I also transform the circular error bounds into error ellipses at the end of this chapter.

Both circles and ellipses can be encoded as three standard deviations of a normal distribution using the covariance matrix for the estimated state (x, y). The error radius directly determines the operational capability of the autonomous vehicles, thus must be accurately represented.

I model three components that contribute to the error radius:

1. Error in the camera's own location and orientation due to imprecise installation
2. The algorithmic error in the estimation of the pixel that represents the centre of the vehicle
3. The ground area covered by the pixel that represents the centre of the vehicle (encoding cameras' finite precision)

These will be analysed in turn. Note that I assume calibrated cameras without distortions, and no motion blur.

4.1.2 Location and Orientation Error

Of the three sources of error, this one is the most difficult to reason about. Many transport authorities have standard mounting specifications, including

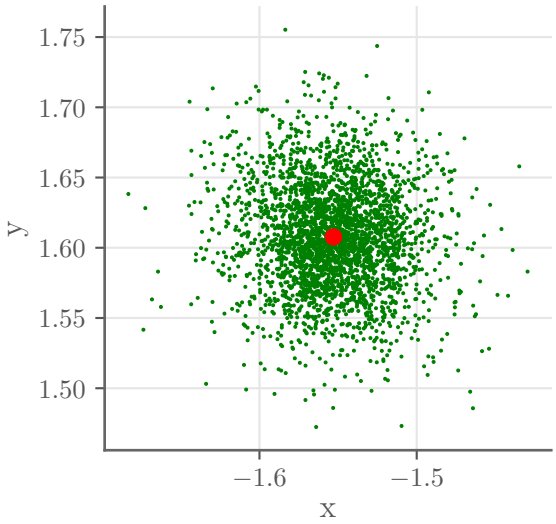
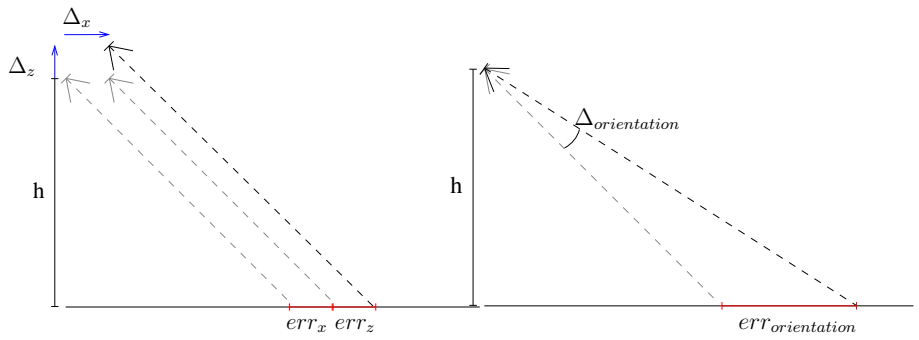


Figure 4.1: The red point is a true world location, while green points were estimated by a large sample of cameras, incorporating positional, orientational and algorithmic errors. We must associate an error with each green point.

mount heights, for cameras [36]. However, none of these specify the tolerances in location and orientation, and we will have to make some educated guesses.

I assume that the position and orientation error is normally distributed, with 95% of positions within σ_{xyz} meters in any direction, and orientation within a $\sigma_{orientation}$ degree conical radius about the principal axis 95% of the time. In mathematical terms, this means Δ_x, Δ_y, and Δ_z are all sampled from $\mathcal{N}(0, \sigma^2_{xyz})$. The orientation error is given as $\Delta_{orientation} \sim \mathcal{N}(0, \sigma^2_{orientation})$.



(a) Positional Errors (b) Orientational Error

Unless otherwise mentioned, I throughout this report default to an installation accuracy of $\sigma_{xyz} = 0.015m$ and $\sigma_{orientation} = 0.25°$. These relatively tight bounds could be achieved using a post-installation calibration.

To translate these values into an uncertainty on the ground plane, consider that any horizontal camera movement (Δ_x, Δ_y) from its true position simply translates the predicted vehicle location and therefore increases its error by the length of the translation. I call this combined value err_{xy} and at two standard deviations it is bounded above by $|err_{xy}| = 2\sigma_{xyz}$.

Vertical translation, $\Delta_z \sim \mathcal{N}(0, \sigma^2_{xyz})$ of the camera induces an error as $err_z(d, h) = \Delta_z \frac{d}{h}$ where d is the ground distance to the vehicle (on the XY plane), and h is the true intended height of the camera.

Thus, the total contribution from the camera's positional error to the error radius will (95% of the time) be bounded by $(err_{xy} + err_z(d, h))$, which evaluates to

$$err_{xyz}(h, d) = 2\sigma_{xyz} + \frac{2\sigma_{xyz}d}{h}$$

Similarly, it can be shown that using a conical error region about the desired principal axis of magnitude $\Delta_{orientation} \sim \mathcal{N}(0, \sigma^2_{orientation})$, for a camera at height h, and ground distance to the vehicle d, the error induced in the estimation of the vehicle's centre is

$$err_{orientation}(h, d) = h \tan\big(\tan^{-1}(d/h) + \Delta_{orientation}\big) - d$$

4.1.3 Algorithmic Error

The vehicle localization algorithm needs to scan the pixel plane for vehicles, and return pixel representing the centre of vehicles on the ground plane. The centre of the pixel projected onto the ground plane is taken as the vehicle centre.

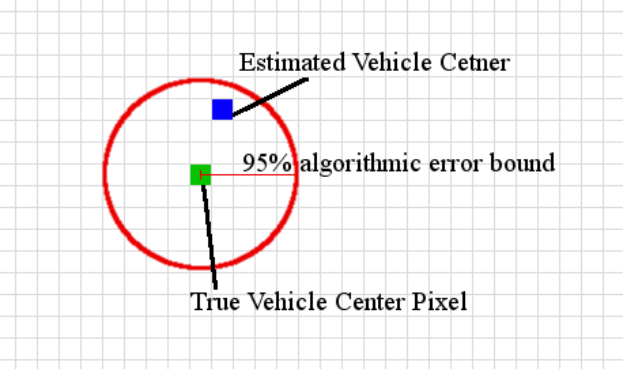


Figure 4.3: Illustration of Algorithmic Error in the camera's pixel plane. The radius is a 95% bound with length $2\eta_{alg}$

I assume that at the time of deployment, computer vision algorithms are very good at this task, but may still be wrong by several pixels. I analyse the effect of this inaccuracy in the model validation Section 4.2 and refer to the Euclidean distance in the pixel plane from the true centre pixel to the calculated one as $\Delta_{alg} \sim \mathcal{N}(0, (\eta_{alg}^2)$. Unless otherwise specified, I default to a 95% radius of 4 pixels, setting $\eta_{alg} = 2$.

4.1.4 Error from Camera Limitations

Lastly, the error induced by the camera must be accounted for. Consider that in the limit, a vehicle is covered by exactly one pixel, meaning any estimate of its position is at least as uncertain as the ground area covered by that pixel. On the other hand, if a vehicle exactly fills the camera plane and the localization algorithm perfectly determines the centre of the vehicle in the pixel plane, the estimate still cannot be better than the area covered by the single pixel.

I call the area covered by a pixel p, $A(p)$. $A(p)$ is also a function of the camera position and orientation, and the camera parameters (field of view and resolution), but these are omitted here as they are constant for any given camera.

Correspondingly, an approximate upper bound on the longest distance within

a pixel can be calculated as the diagonal of a square, resulting in a ground distance of $err_{pixel}(p) = \sqrt{(2A(p))}$.

4.1.5 Complete Error Model

I now combine the three sources of error to determine an approximate error radius for the vehicle localization.

$$r(h, d, p) = err_{xyz}(h, d) + err_{orientation}(h, d) + \Delta_{alg} * err_{pixel}(p)$$

For any individual camera, the err_{xyz} and $err_{orientation}$ terms introduce a directed bias for any point in the pixel plane, though over many (slightly mis-)installed cameras these terms have a normal distribution. The Δ_{alg} is modelled as normal as well. The err_{pixel} term is a systematic and complex function of camera parameters and estimated vehicle location, and the only component with a non-zero mean over a large number of samples.

To use r in simulation, when a vehicle is visible to a camera, I project the ground truth vehicle centre to a (randomly slightly-mispositioned) camera, automatically incorporating the positional and orientational errors. The algorithmic error shifts the incident pixel a distance sampled according to the distribution of Δ_{alg}, in a randomly sampled direction. Finally, I use a second, fake, perfectly positioned camera to calculate the pixel error and reproject into a world position from the perturbed pixel.

4.2 Error Model Validation

In this section I evaluate the error radius function developed above. It is a good error model if,

1. For a given estimated vehicle location, the true location should be within the predicted error radius most of the time. My formulation

aims for 95% accuracy.

2. It is not over-conservative: the tighter the bound, the more useful the update is for vehicle navigation.

I also examine how the different components that make up the error radius vary as the pitch of the camera varies, followed by a brief look at the effect of algorithmic inaccuracy. Experiments were performed with the camera placed at a height of 6 meters above the ground, and a fixed field of view of 60 degrees horizontally and vertically. Rotation about the Z axis was ignored as it does not affect results.

4.2.1 Quality of r as an Error Bound

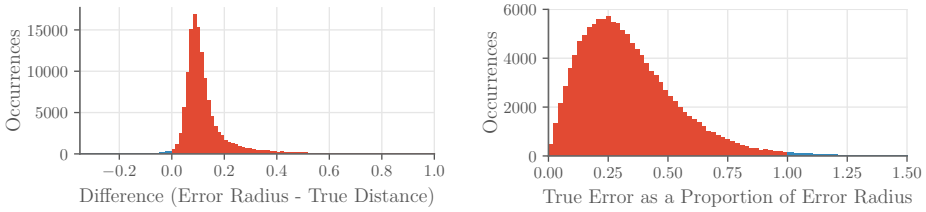
For this and the following subsection I assume a perfect vision algorithm, setting Δ_{alg} to 1, and use a camera with resolution 400x400 pixels and field of view of 60 degrees horizontally and vertically.

Table 4.1: Success Rate of r as an Error Bound

Total Samples	# Within Error Bound	% Within Error Bound
125000	124290	99.432

Table 4.1 shows that more than 99% of 125000 sampled camera positions and vehicle positions (randomly placed to account for installation inaccuracy), the distance from the predicted vehicle position to its actual position is within the calculated error radius. This is actually better than intended, since the construction was for 95% accuracy.

It is easy to achieve an error bound that is always valid: simply set it very high. Figure 4.4a shows two important characteristics. Firstly, the difference between the predicted error radius and the actual error distance is small, meaning it is not an excessively high bound. Secondly, when the bound is *incorrect* (very small region less than zero in blue) the true error only slightly exceeds the error bound.



(a) Computed radius minus actual error distance.

(b) Actual error distance as a proportion of radius.

Note that the actual error is dependent on several random variables – sampled positional and orientational error, and perturbations modelling algorithmic error (pixel ground area is not randomly sampled). Each of these are normally distributed and added together (in absolute values). Since the computed error bound is in the tail of this distribution, we should expect that most of the time, it is actually several times too large. I plot the true errors from Figure 4.4a as a proportion in Figure 4.4b, showing what fraction of the bound the actual error consumes.

As expected, most samples are a small fraction of the bound. Attempts to tighten it resulted in the loss of the 99% accuracy. This doesn't look entirely like a rectified Gaussian because of the square root operation in Euclidean distance.

Overall these results suggest the r is good at modelling the true error, even though during construction I did not model interplay between some of the factors (eg. how orientation or positional error affects pixel area inaccuracy).

4.2.2 Error Radius as a function of Pitch Angle

The XY offset of a camera induces a constant distance on the ground plane. However, all other components modelled in the error bound are dependent on pitch angle. Notably, pixel areas diverge as the camera angle approaches horizontal. Figure 4.5 illustrates the components of the error radius as a function of pitch.

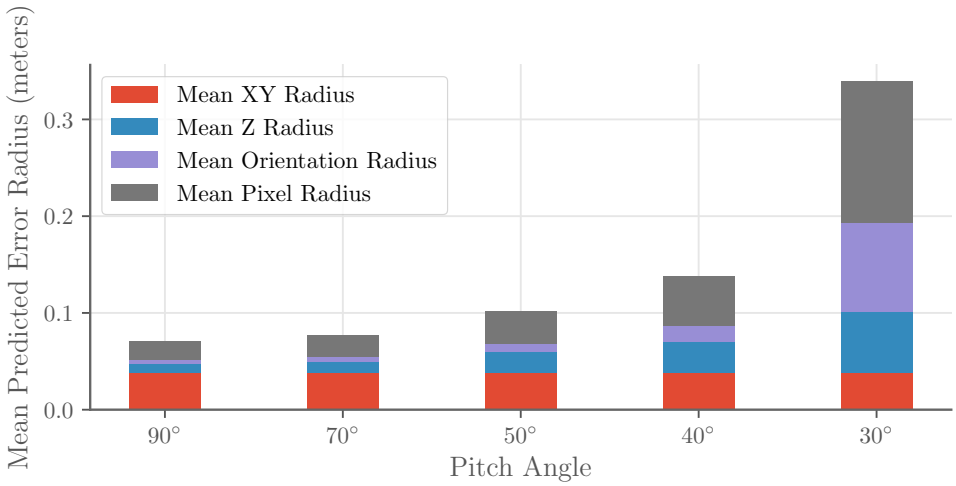


Figure 4.5: How r varies as a function pitch angle. Error bars are left out for clarity.

This gives us an intuition for how each factor impacts the radius. As expected, orientation and pixel area contributions come to dominate, while at 90°, when the camera is pointing straight down, the dominant error is the camera's horizontal translation.

Also note that the pixel-area component of the error is the only component that can be affected by the camera's configuration and the algorithmic accuracy in detecting the centre pixel of the vehicle. This leads to some fundamental limits discussed in Section 4.3.

4.2.3 Impact of Algorithmic Inaccuracy

The prior subsections were all presented without any algorithmic inaccuracies. However, no computer vision algorithm can be expected to perfectly determine the pixel containing the centre of the vehicle.

Figure 4.6 shows how the mean error radius changes as a function of the algorithmic error in red. In blue, I show that the accuracy as a bound is minimally influenced by the introduction of the extra error source. This implies that my bound of algorithmic error is indeed well matched to the

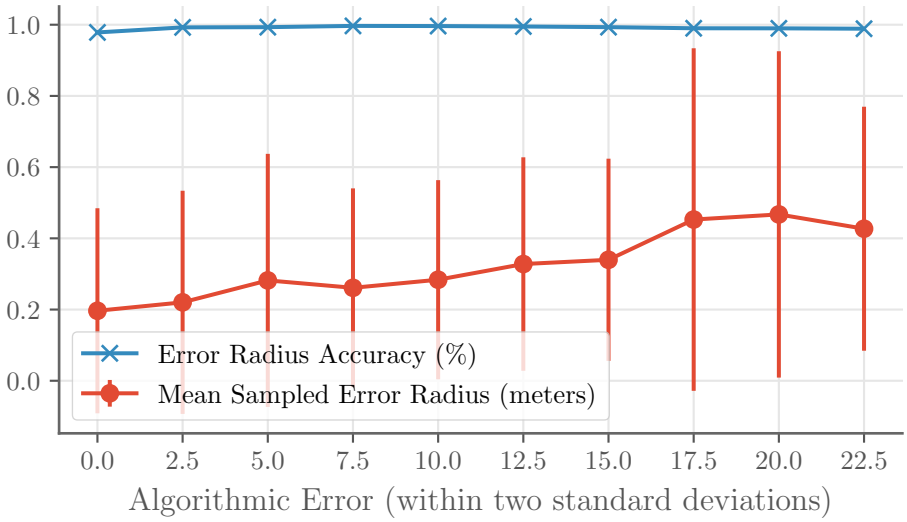


Figure 4.6: The effect of algorithmic inaccuracy on the mean value and accuracy of r. Error bars indicate two standard deviations.

growth of the actual error.

4.3 Limits and Implications

Having validated that the error model serves as a good bound on the real errors, we can use the function r to reason about fundamental limits of the proposed system.

4.3.1 Effect of Resolution

One interesting question to ask is: if we had infinite resolution, how well could we perform? Alternatively, how much accuracy do we gain from increasing resolution?

Figure 4.7 shows that the mean error bound stops decreasing significantly beyond a resolution of 2000 by 2000 pixels. A smaller pitch angle gains further from higher resolution, which is in line with what we expect from

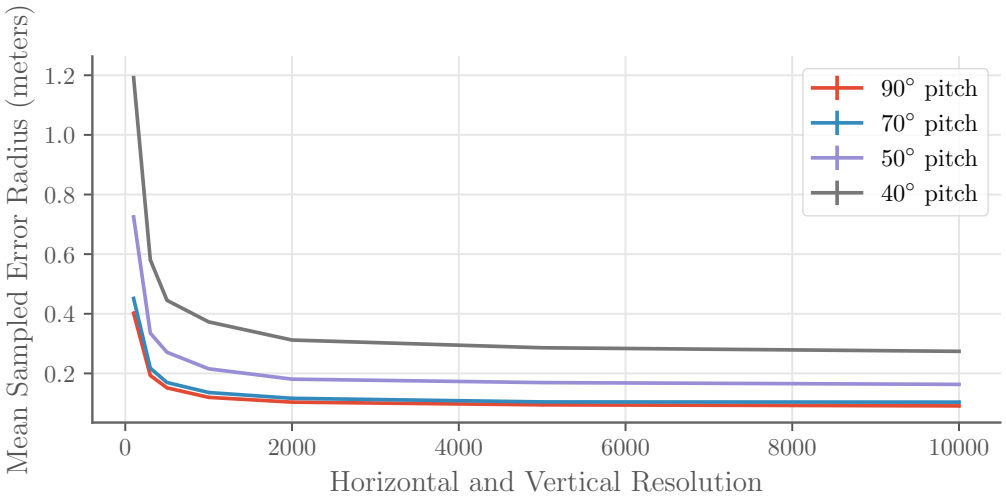


Figure 4.7: How increased resolution can increase the error bound at different pitch angles. Error bars left out for clarity.

Figure 4.5 where we saw that more horizontal cameras have a larger part of their error bound stemming from pixel area uncertainty, which can in turn be reduced further by increasing resolution.

A similar behaviour can be had by decreasing the field of view, which was for these experiments set to 60° horizontally and vertically. In the next chapter, I fix the resolution and only optimize for the field of view, since a higher resolution always increases the performance of the proposed system at no cost and is thus not interesting.

4.3.2 Installation Requirements

Figure 4.7 was created assuming the following 95% bounds: positional error within 3 centimetres in any direction, orientational error within a 1° diameter cone about the intended axis, and an algorithmic inaccuracy radius of 4 pixels. Table 4.2 shows how well such a configuration might perform at different pitch angles (sampled from a large number of cameras instantiated according to the allowed variations, and across various world vehicle positions).

Table 4.2: Sampled Prediction Error Bounds using 2000x2000 resolution

Pitch	**Mean (m)**	**Median (m)**	95^{th} **Percentile (m)**
90°	**0.1035**	**0.1035**	**0.1172**
70°	**0.1163**	**0.1075**	**0.1658**
50°	**0.1824**	**0.1454**	0.3866
40°	0.3167	**0.1884**	1.022

As discussed in Section 1.2, allowable lateral deviations are generally quoted at ±10-25cm for autonomous vehicles. The table above shows in bold which camera placements would be within these limits. Note that even though the mean and 95% percentile error bounds at lower pitch angles are high, the median tells us that most measurements from such a camera would still be useful.

Next, I present some installation accuracies that would need to be achieved in order for the system to provide acceptable performance. In bold in Tables 4.2, 4.3, and 4.4 I highlight positional and orientational constraints that provide sub-20 centimetre median localization performance.

Table 4.3: Median Localization Bound for 70° pitch, 2000x2000 pixels

	Orientation Error, 95%							
95% Pos. Error (m)	0°	0.2°	0.4°	0.6°	0.8°	1°	1.2°	1.4°
0.0	**0.017**	**0.034**	**0.051**	**0.069**	**0.087**	**0.103**	**0.12**	**0.15**
0.1	**0.177**	**0.196**	0.214	0.230	0.248	0.268	0.28	0.29
0.2	0.342	0.360	0.378	0.394	0.409	0.430	0.44	0.46
0.2	0.503	0.524	0.538	0.552	0.566	0.586	0.61	0.61
0.4	0.665	0.685	0.702	0.715	0.737	0.757	0.77	0.79

These tables show that achieving a 10 centimetre installation accuracy 95% of the time is an absolute necessity, while angular deviations of up to one degree are probably acceptable. Finally, Table 4.5 shows how algorithmic inaccuracy impacts localization: with the usual 3 centimetre placement and

Table 4.4: Median 99% Localization Bound for 45° pitch, 2000x2000 pixels

95% Pos. Error (m)	**Orientation Error, 95%**							
	0°	0.2°	0.4°	0.6°	0.8°	1°	1.2°	1.4°
0	**0.027**	**0.060**	**0.083**	**0.114**	**0.145**	**0.180**	0.21	0.23
0.1	0.222	0.250	0.285	0.324	0.338	0.365	0.39	0.44
0.2	0.421	0.460	0.484	0.504	0.540	0.561	0.60	0.63
0.3	0.624	0.661	0.682	0.713	0.761	0.764	0.81	0.84
0.4	0.839	0.828	0.873	0.921	0.950	0.971	1.03	1.03

a 0.5 degree orientation error as well as 45 degree pitch. The algorithm must be able to localize the vehicle centre to within 10 pixels at the most.

Table 4.5: Median Localization Bound for 45° pitch, 2000x2000 pixels with increasing algorithmic errors.

Error (pixels)	4	8	12	16	20	24	28
Median r (m)	**0.157**	**0.183**	0.219	0.237	0.260	0.294	0.313

In the analysis above I used a tighter bound for localization error (20cm); some literature allows for up to 40cm lateral deviation, and it is unclear if this is total, or in either direction. The values tabled here are radii, so they can be in either lateral direction. Tighter bounds can be achieved if the algorithmic performance is very good or installation is extremely precise.

4.4 Error Circles to Error Ellipses

The prior sections used circular error bounds, which while also accurate and insightful, can be slightly improved upon. A quick inspection of position estimates versus their true positions in Figure 4.8 shows that at higher pitch angles, the estimates lie along an oval rather than a circle.

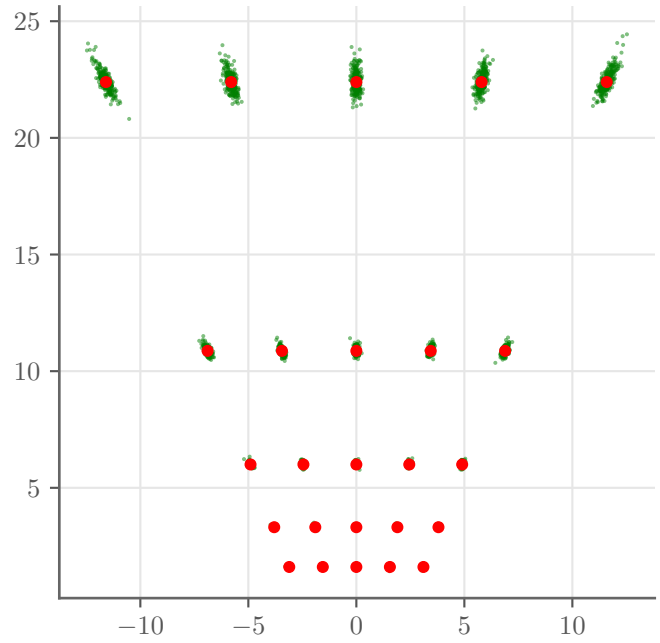


Figure 4.8: Ground plane estimates (green) of true position (red) across a large sample of cameras showing algorithmic, positional, and orientational error.

I thus reshape the error circles into ovals of equal area, with the major and minor axes determined by the ratio of ground height to width of the chosen pixel. The orientation of the ellipse is given by the angle from the camera to the pixel's ground position.

This modification improves the accuracy of the error bound even further, as shown in Table 4.6. To see why this occurs, in Figure 4.9 I show both circular and elliptical error bounds. One case where the circular error bound fails while the elliptical one succeeds is shown in bold.

Table 4.6: r as an Elliptical versus Circular Error Bound

Bound Type	# Samples	# Within Error	% Within Error
Circular	125000	124290	99.432
Elliptical	125000	124528	**99.622**

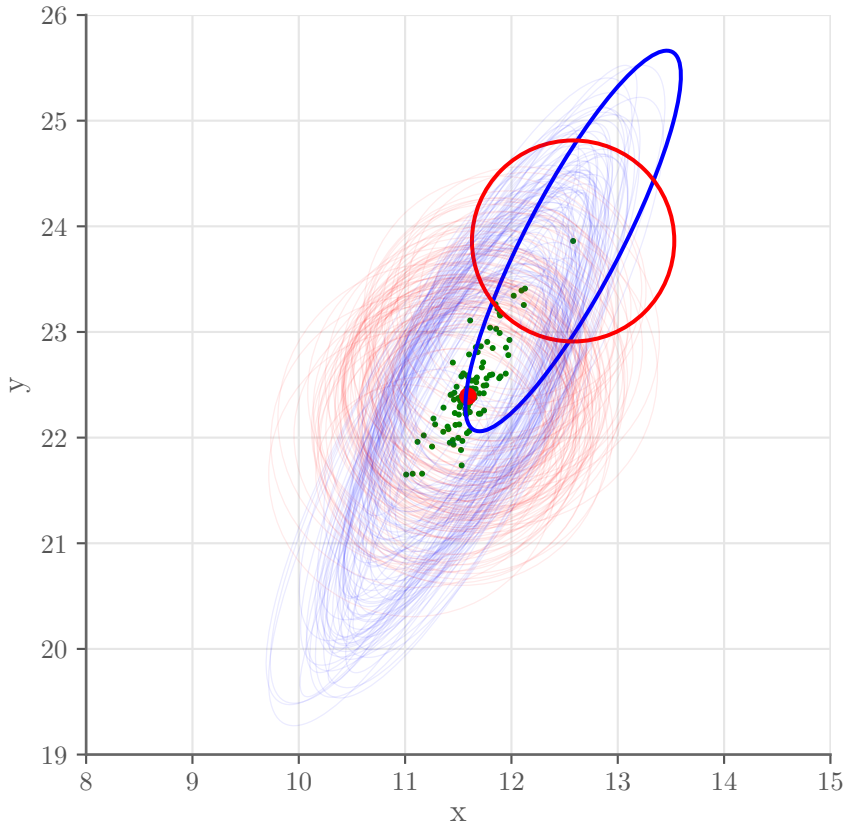


Figure 4.9: An example of where elliptical error bounds are more accurate than circular ones. Red circles represent error circles, while error ellipses are in blue. Note the vast majority of both are accurate bounds (very lightly drawn ovals and circles)

In simulation I encode oval error bounds defined by the angle ϕ and length of major axis M, and minor axis N into the covariance between x and y accoring to the following formula:

$$\Sigma_{x,y} = \begin{bmatrix} M^2 \sin(\phi)^2 + N^2 \cos(\phi)^2 & (M^2 - N^2) \sin(\phi) \cos(\phi) \\ (M^2 - N^2) \sin(\phi) \cos(\phi) & M^2 \cos(\phi)^2 + N^2 \sin(\phi)^2 \end{bmatrix}$$

4.4.1 Practical Issues

In practice, I found that due to timing delays between parts of the simulation, the camera update delivered to the vehicle was slightly old compared to the vehicle's actual state (much as it might be in a real implementation). This difference ended up having a big impact on the EKF update step, and even more so if an oval was used (when only 1 dimension was stale, the stale value would pollute the remaining good value). I fixed this using the camera update's timestamp to extrapolate the missing interval, which worked quite well in practice and could be transferred to real world implementations. More complex methods for handling delayed measurements can be found in [5][13].

4.5 Summary

This chapter has developed and evaluated the camera sensor model used in the following chapter to provide correctional updates to an autonomous vehicle. To achieve this, the centre of the vehicle needs to be estimated in world coordinates. This is associated with an error, which informs how trustworthy an individual update is.

The error bound developed here, r, is a function of camera height, pixel area on the ground, and the horizontal distance to the vehicle centre. It was shown to be an effective upper bound, 99% of the time, and small enough to be usable in a real-world implementation (assuming the installation bounds can be achieved). The effect of pitch angle was extensively analysed, and Section 4.3 explored the impact of resolution in the limit, providing a fundamental limit on the effectiveness of the proposed system, as well as some real-world requirements in terms of installation and algorithmic accuracy. Finally, I showed the effectiveness of reshaping circular bounds into elliptical ones.

Chapter 5

Camera Placement

Two of the questions this dissertation sets out to answer are related to the placement of cameras for localization and navigation. Normally, problems of this type are formulated one of two ways. Either, given a budget of n cameras, find the best placement according to some objective function, or given some objective function criteria, place as many cameras as required. I focus on the first formulation in this chapter.

5.1 A Cheap Heuristic Objective

Since I hope to explore up to four degrees of freedom in a single camera placement (pitch, yaw, horizontal and vertical field of view), the search space is even larger than what previous authors in this field have dealt with. Additionally, my baseline metric based on prior work, mutual information, is evaluated using Monte Carlo approximation, an extremely expensive computation. In my experiments I found that about 2000 simulations were required to achieve stable results over repetitions of the same experiment.

Due to its cost and the size of the search space, I attempt to develop a cheaper objective that can be used for optimizing the parameters of a single camera. Some high scoring parameters from this search are then used as discretised

points in the full optimization.

5.1.1 Objective Function for Localization

The off-board camera network aims to maximally inform vehicle localisation: a natural objective is therefore to maximise the reduction in vehicle uncertainty after an update.

Peeking into the implementation of the Kalman filter, we can deduce that maximal reductions in vehicle uncertainty occur when we receive maximally accurate updates from the offboard cameras. Since the covariance of an update is dependent on r from the prior chapter, we may consider minimizing the expected value of r reported by a camera.

Consider then that we know roughly where vehicles will be: for vehicles to be operating they must be within 25cm lateral error of the target path. Assuming the steering controller is efficient, this implies the robot's belief of its own position is also within 25cm of the path. Thus, assuming the 25cm represents two standard deviations, the vehicle will be within 25cm of the path 95% of the time.

Using this information, I place a normal distribution along the path, which defines the probability of being at a distance from the centre line at all points, and call it a function P.

So, for a camera C out of all possible configurations $Cameras$ at the given position, camera to ground projection function $proj_C(i, j)$ from pixel (i, j), and error radius function $r(i, j)$ (shorthand for the function in Section 4.1.5) I formulate:

$$argmin_{C \in Cameras}(obj_1(C)) \tag{5.1}$$

$$obj_1(C) = \sum_{(i,j) \in C} P(proj_C(i, j)) * r_C(i, j) \tag{5.2}$$

The problem with the given objective function is that a minimum is achieved when the camera does not see the road at all, driving the expectation to 0. Really, the problem needs to be reformulated as a maximisation. To do this, r needs to be inverted, but there are many ways to achieve this, including negation and inversion.

My insight is to utilize the Kalman gain, $\boldsymbol{K}$, from Section 3.4.2. It directly quantifies how much the vehicle's localization will "trust" an update, and is highest when the error in the measurement is low. This provides a principled way of inverting r and generating a maximisation problem.

A simple one dimensional Kalman Filter computes the gain:

$$K = \frac{\sigma^2}{\sigma^2 + \sigma^2_{measurement}} \tag{5.3}$$

So, directly using r as the measurement error, I set $\sigma = 1$ and $\sigma_{measurement} = (r/3)$, resulting in the following objective function:

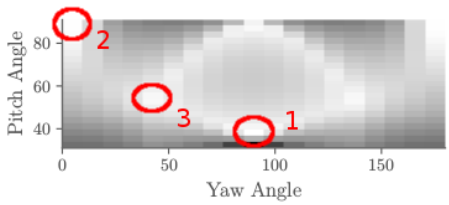
$$obj(C) = \sum_{(i,j) \in C} \frac{p(proj_C(i,j))}{1 + (r_C(i,j)/3)^2} \tag{5.4}$$

This expectation can cheaply be evaluated across the pixel plane of the camera, and is even well approximated with resolutions as low as 30x30 pixels. Thus, it is amenable to exhaustively search.

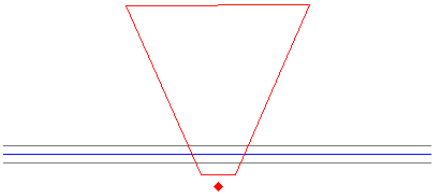
5.1.2 Optimizing Pitch and Yaw with varying Curvature

I perform a grid search over the pitch and yaw parameters using the defined objective function, fixed field of view at 60 degrees in both directions, and a camera at a height of 6 meters. The results are presented in Figure 5.1.

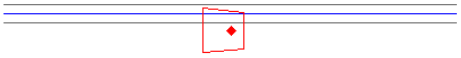
The first images in a row show the evaluated objective function obj, the others are diagrams of the path and camera's field of view. The displayed configurations are the top-scoring and distinct orientations (I ignored sym-



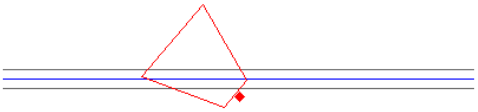
(a) 0 Curvature – lighter is better



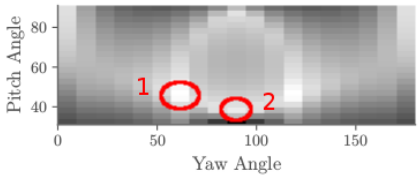
(b) Configuration #1, top score



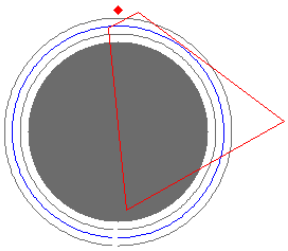
(c) Configuration #2



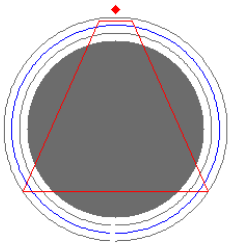
(d) Configuration #3



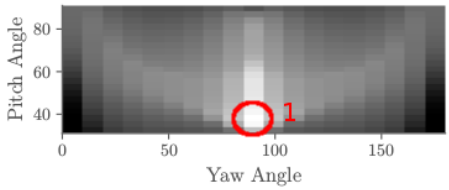
(e) 20m Radius



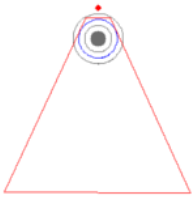
(f) Configuration #1



(g) Configuration #2



(h) 5m Radius



(i) Configuration #1

Figure 5.1: Results and high scoring orientations from the cheaper objective function. Lighter colours indicate higher objective values.

metric cases). The filled grey areas are world obstacles that I place to imitate walls.

The presented optimal points match what one might intuitively expect for camera placement: we maximise both the length of the path we can spot

at once and have the path trace through the highest pixel density in the camera's field of view – this gives the lowest error during location prediction.

As the curvature increases, the number of different "good" orientations decreases – with a highly curved path, there is only one best choice - the one observing the most of the path possible.

Going forward into the more expensive objectives in Section 5.3, I use the points on the ridge of "good" values (bright colours) as allowed orientations.

5.1.3 Optimizing Pitch, Yaw, and Horizontal and Vertical FoV Simultaneously

As mentioned in the introduction of this dissertation, I also attempt to optimize the field of view simultaneously with the pitch and yaw directions, which has not been examined as a variable before.

To examine this I used a hill-climbing optimization algorithm [33] with random restarts to explore the space of options, as the search space is now $O(n^4)$ rather than $O(n^2)$ and makes exhaustive search prohibitive.

Hillclimbing – Straight Road

For straight roads, all of the top scoring configurations shrunk the field of view as far as was allowed, and oriented themselves to view the road along the diagonal of the field of view. This makes sense – when observing a straight line, there is no point wasting pixels on unlikely regions. Additionally, shrinking the field of view is akin to increasing the resolution, and decreases the error of predictions.

Scoring significantly worse are more a few diversified choices, as depicted in red and purple in Figure 5.2.

Table 5.1: Three top scoring, diverse results from hillclimbing across four parameters on straight roads.

Objective Score	Pitch	Yaw	H. FoV	V FoV.
8.63 (green)	57	90 ± 43	20.1	20.1
...				
6.6787 (red)	68	90	47.2	22.9
...				
3.0710 (purple)	39.3	90	25.1	53.5

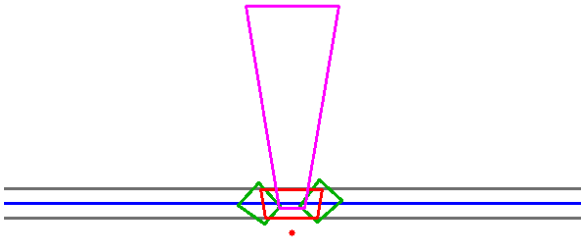


Figure 5.2: Visualization of Top Hillclimbing Result for straight roads, each colourful trapezoid is a specific camera configuration's view.

Hillclimbing – Curved Roads

We can obtain slightly more consistently varied results when optimizing on curved roads. Although the top two scoring parameters still minimized the field of view, the next best result had a horizontally wide but vertically narrow field of view. Compared to straight roads, we see more diverse solutions with high scores.

Table 5.2: Top 4 results from hillclimbing across four parameters with curvature 0.05 (symmetric possibilities are colour coded the same).

Objective Score	Pitch	Yaw	H. FoV	V FoV.
8.58 (green)	56.5	270 ± 25	21.1	20.4
7.98 (red)	61.0	270	98.2	22.1
...				
3.72 (purple)	43.3	270 ± 45.0	38.8	50.8

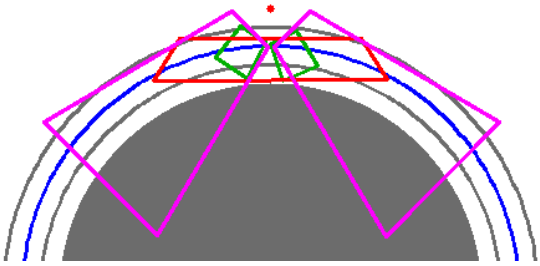


Figure 5.3: Visualization of Top Hillclimbing Result for a curvature of 0.05.

5.1.4 Discussion

The objective function used in this section can be seen as a measure of the *instantaneous localization capability* of the camera, not measuring actual vehicle's performance but expected performance along a road. In other words, if a vehicle appeared suddenly, or could only be sent a single update, this metric captures how the camera configuration should be chosen to reduce its localization error, on average.

There are several issues with this metric. Biggest of them is the embedded assumption that only 1 update can be sent. In reality, a vehicle spends a longer amount of time in the camera's field of view if the camera can cover more ground area – though the resulting updates sent to the vehicle may be less accurate. A useful property of the Kalman filter is that the uncertainty (covariance) in the state can only shrink as a result of a measurement, so there is some utility in sending more updates over a wider field of view, even at reduced accuracy.

The advantages of the objective function here are twofold: firstly, it is cheap to compute, and is the reason I could do the hillclimbing optimization which required hundreds of evaluations of the objective. Secondly, it can be used independently of specific trajectories; all other objective functions in this work are centreed about a single path. The latter property is discussed as a future research direction at the end of this report.

5.2 Navigation-specific Metrics

Here I define more tailored metrics that I use to evaluate camera placements. These address the limitations of my heuristic such as assuming only a single update can be sent. By utilizing the vehicle simulation to evaluate the metrics, other factors such as vehicle speed, controller performance and orientation of the covariance ellipses are also considered.

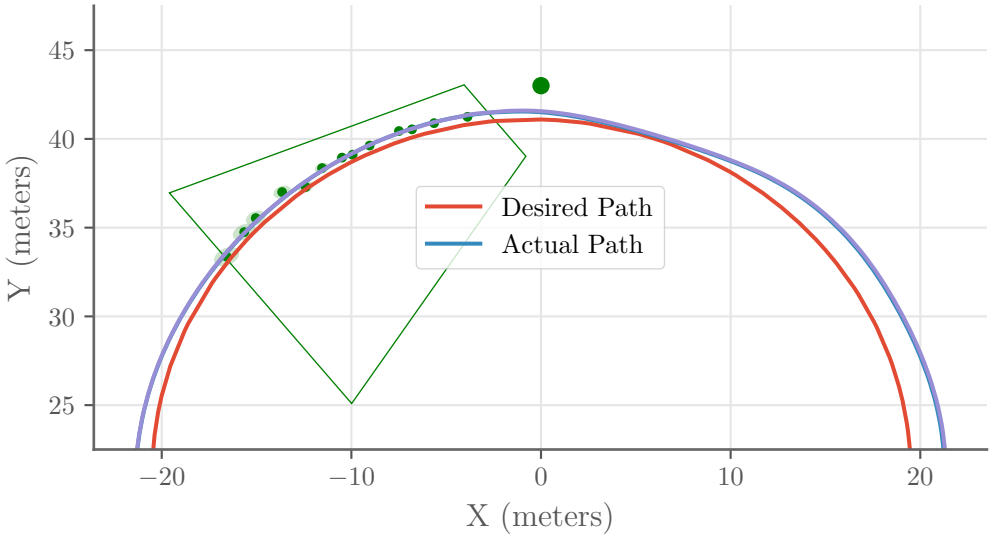


Figure 5.4: Example of a camera sending updates to a vehicle tracking a path, with errors as small green ovals. The vehicle corrects its positional estimate after each update.

In this section, x_t denotes the robot state at time t, u_t denotes the control command that drives the odometry motion model from x_{t-1} to x_t, and z_t^A is the measurements received at time t from cameras in the set A. Corresponding capital letters are used to denote the random variables for state, controls, and measurements.

5.2.1 Mutual Information

I use as my objective functions the one from [22], and describe it following their notation. The goal is to maximise the mutual information between all possible robot states $X_{0:T}$ on a path, and the observations of the robot along the same path from a set of cameras A, written as $Z^A_{1:T}$, given the robot controls $U_{1:T}$.

Formally, this is written as

$$I(X_{0:T}; Z^A_{0:T}|U_{0:T}) = h(X_{0:T}|U_{0:T}) - h(X_{0:T}|U_{0:T}, Z^A_{0:T})$$

The first term is the joint entropy of all possible robot states given all possible controls (please see Section 3.5.1 for definitions of entropy). This value is constant for a path. So, maximising mutual information is equivalent to minimizing the conditional entropy of robot states given the observations and controls.

For continuous variables we can compute $h(X_{0:T}|U_{0:T}, Z^A_{0:T})$ using the following integral (I drop the subscripts for convenience).

$$-\int\int p(x|u, z^A) log p(x|u, z^A) dx p(u, z^A) d(u, z^A) \tag{5.5}$$

Evaluating this integral in closed form is not possible, so I use Monte Carlo evaluations to sample from the distribution of controls and measurements.

The last missing piece is to obtain the joint distribution of all robot states $p(x_{0:T}|u^A_{1:T}, z^A_{1:T})$. This is often rewritten as:

$$p(x_{0:T}|u^A_{1:T}, z^A_{1:T}) = \\ p(x_0) \prod_{t=1}^{T} \eta_t p(z^A_t|x_t) p(x_t|x_{t-1}, u_t) \tag{5.6}$$

This breaks down the joint posterior distribution of all robot states into the

product of the motion model $p(x_t|x_{t-1}, u_t)$ and the sensor model $p(z_t^A|x_t)$ at each time step. I use the sensor model developed in Chapter 4 to obtain approximate probabilities of getting a concrete sensor measurement, given the robot's mean positional belief.

Lastly, the η normalizer needs to be computed. It can be shown to be equal to:

$$\eta_t = \int p(z_t^A|x_t)p(x_t|u_{1:t}, z_{1:t-1})dx_t \tag{5.7}$$

the first term is again the sensor model, and the second is recognizable as the belief of the robot, given by the mean and covariance. I evaluate the normalizing constant with another small Monte Carlo approximation, drawing sample states x_t from the robot's positional distribution.

The joint probability distribution can be seen as the probability that the robot took the exact path that it did on any given execution of the robot's path. Since these are all drawn from continuous distributions, technically any such probability should be zero. I discretise using a small delta when calculating the probabilities to obtain small but finite values. However, this also requires that the Monte Carlo approximation be run several thousand times to obtain valid results.

5.2.2 Mean Trace of Covariance Matrix

Alternatively, I consider another measure of uncertainty: the trace of the covariance matrix. I use the mean trace across T timesteps in a single trajectory:

$$MTrace = \sum_{t=0}^{T} tr(\Sigma_t) \tag{5.8}$$

The trace of the covariance is mathematically related to the sum of the radii of the confidence ellipse, so this objective has an intuitive interpretation.

5.3 Placing Single Cameras With Navigational Metrics

I present selected results from multiple experiments with the mutual information and mean trace as placement metrics. Due to the size of the search space and cost evaluating the metrics, greedy optimization is used. In past work, submodularity was used to justify this choice, which provides constant-factor bounds on the greedy solution to particular NP-hard problems. Unfortunately, my problem misses this property, and consequences are discussed in Section 5.5.

5.3.1 Straight Roads

The first experiment minimized conditional entropy (identical to maximising mutual information) to compute the best orientation of a single camera on a straight road, with a fixed field of view.

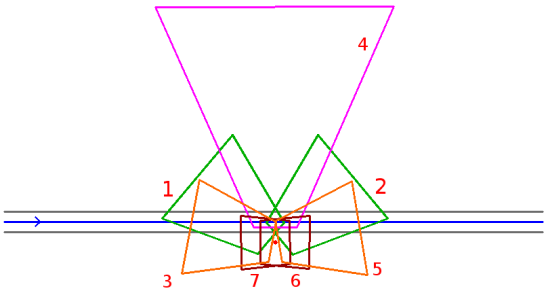


Figure 5.5: Visualization of camera orientations and rankings.

The two highest scoring orientations are the diagonal orientations, labelled 1 and 2 in Figure 5.5. These were among the middle of the range of predicted orientations by the heuristic. Table 5.3 ranks the choices and shows some other possible measures, such as the lateral error and total trace.

The trace and mutual information measure generally agree on placements, but don't necessarily imply the lowest lateral error from the path.

Table 5.3: Mean values of various metrics for each allowed orientation. Vehicle travels left to right. Results from 2000 simulations with a camera at 6 meters elevation.

Rank	$h(X\|U,Z)$	Total Trace	Lateral Error	Localization Error
1	**4e-549**	**1.788**	**0.0151**	0.01382
2	5e-546	2.05	0.0163	**0.01381**
3	6e-540	2.14	0.0168	0.01391
4	2e-537	2.11	0.0165	0.01394
5	9e-537	2.16	0.0154	0.01443
6	2e-536	2.18	0.0174	0.01646
7	5e-534	2.28	0.0165	0.01683

I also used results from the hillclimbing joint optimization of field of view and orientation, which I visualize and rank in Figure 5.6.

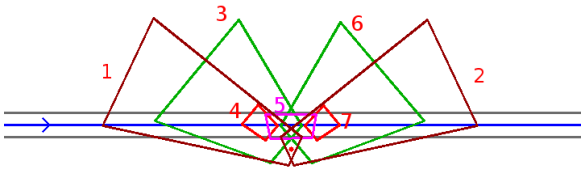


Figure 5.6: Visualization of camera orientations and rankings.

The result of this evaluation yielded some interesting results. Firstly, the hillclimbing algorithm from Section 5.1.3 consistently decreased the field of view to obtain a better score – that seems to be detrimental for mutual information and covariance trace measures. Even a wide field of view along the path (rank 5, purple) scores badly in these respects. This stems from being able to send multiple updates to the vehicle. I also purposely chose to provide symmetric choices in the optimization process. Here labels 1 and 2 are identical in parameters except being flipped about the perpendicular to the road. In all of the symmetric cases, the optimization ranks the left-facing option higher. This is because facing the vehicle's oncoming direction reduces the vehicle's uncertainty early in the navigation, decrease all of the objective metrics. This is a trend we will continue to see in the following

examples.

5.3.2 Curved Roads

Following on from the heuristic optimization, I test the five options found by the hillclimbing algorithm previously.

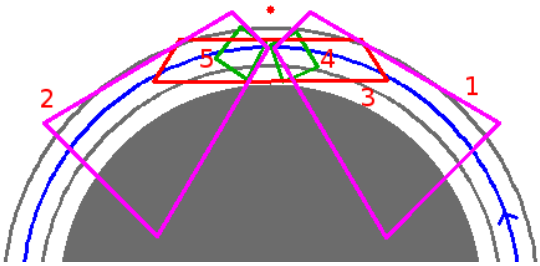


Figure 5.7: Rankings of various options found by the hillclimbing algorithm for a curved road with 20 meter radius.

Once again we see that the views further down the road perform better, as well as the ones that face the oncoming vehicle. Additionally, we can again notice that the optimal points from the heuristic (smallest green squares, numbered 4 and 5) performed badly in this metric, ranking last according to mutual information, and mean trace.

Table 5.4: Mean values of various metrics for each allowed orientation and field of view. Vehicle travels counter-clockwise. Results from 2000 simulations per orientation with a camera at 6 meters elevation. Road has 20 meter radius.

Rank	$h(X\|U,Z)$	Total Trace	Lateral Error	Localization Error
1	**9e-1430**	**6.28**	**0.4433**	**0.02280**
2	4e-1426	8.05	0.4444	0.02531
3	1e-1406	6.95	0.4654	0.02294
4	8e-1390	8.29	0.4435	0.02671
5	5e-1386	8.77	0.4441	0.02751

5.4 Multiple Camera Placement

I now test the placement of several cameras along a straight path using the mutual information metric. There are three possible locations, each with 5 options for orientation. I keep field of view fixed, as I find the hillclimbing results too unreliable overall. Figure 5.8 shows this graphically.

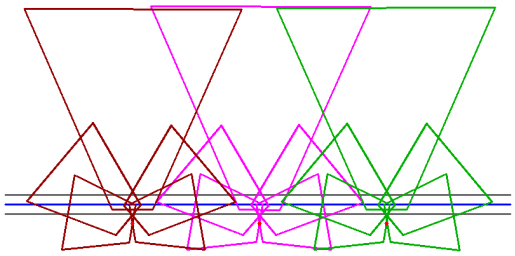


Figure 5.8: on a straight road of length 80 meters, there are 3 possible placement locations with 5 orientations each.

The greedy optimization result, using mutual information, is shown below in Figure 5.9.

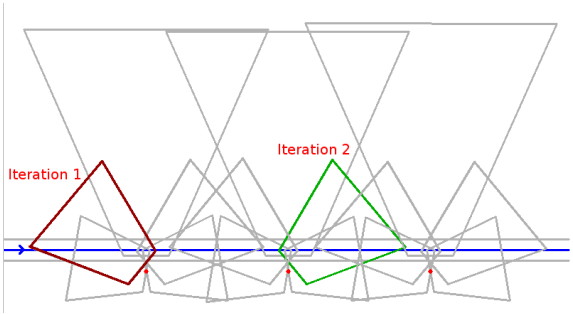


Figure 5.9: The result of greedy optimization of two placements and orientations.

Both mutual information and trace metrics agree with these two placements, by a high margin (selected data is given for the first iteration in Table 5.5). The first choice by the algorithm is the leftmost one, which means there was utility in correcting the vehicle localization early on. Interestingly, the next left-most camera is then skipped, opting for a view slightly further along

the road, and away from the oncoming vehicle. The implication is that the vehicle uncertainty has not grown much, so there is no point spending a camera updating the state when it is still relatively certain. With this property, we would expect to obtain relatively evenly distributed camera placements along a path.

Table 5.5: Top two choices for the first iteration of the multi-camera placement task.

Rank	$h(X\|U,Z)$	Total Trace	Lateral Error	Localization Error
1	**1e-562**	**1.1935**	**0.01393**	**.0181**
2	1e-546	1.43389	0.01461	0.0197
.				
.				
.				

5.5 Non-Submodularity and Implications

None of the metrics presented in this Chapter are actually submodular. Recall that submodularity is the property of diminishing returns – formally, for a discrete set of choices S, take two sets set $A \subset B \subset$. Then pick another, different singleton $c \subset S, c \notin B$. For the objective function F, it must hold over all possible A's, B's and c's that

$$F(B \bigcup c) - F(B) \leq F(A \bigcup C) - F(A) \tag{5.9}$$

In other words, when maximising F, we achieve smaller gains by adding an element to a larger set than to a smaller set. To see why my problem is not submodular, consider that the cameras may have overlapping fields of view. In this case, choosing the second camera will yield only a small improvement in the objective function, while choosing a non-overlapping camera will yield a larger one. Thus, submodularity is invalidated.

If submodularity can be proven, constant-factor optimality guarantees can be

made for NP-hard problems. Although I do not have submodularity, one can very roughly empirically check how submodular an objective is by sampling subsets A, B, and singleton c, and testing how often and by how much the submodularity property fails.

I was only able to empirically test the submodularity of the mean trace of the covariance matrix since it requires fewer simulations to obtain stable results. Unfortunately, this yielded that only 75% of randomly sampled placement sets met the submodularity property, and those that failed sometimes had large increasing gains when adding another placement to the larger set B. I suspect this is due to large overlaps in field of view, though was unable to find the root cause of these violations.

The lack of submodularity implies that the greedy algorithm I used may only be a poor approximation for the true global optimum. Future work may thus be forced to consider more expensive optimization routines.

5.6 Discussion

There are a few conclusions to be drawn. Firstly, that all metrics consistently chose to orient a single camera along the direction of the path, rather than perpendicularly to the road. Further, orientations preferred pointing towards the oncoming vehicle. As long as the vehicle is not already well-localized, this provides the lowest average deviation from the path, localization error, and mean covariance trace.

On the topic of direction, the paths tested were only traversed either from left to right or counter-clockwise. Given more time, it would have been interesting to compute metrics for a conceptual two-lane road that doesn't bias results towards the start of the path.

Ultimately, I found that although mutual information is the more principled measure, the results provided by the mean trace of the covariance matrix largely agreed on the optimal configuration. Even when the two metrics dis-

agreed, their top three choices for the optimal configuration always contained the other metric's top prediction (so, they don't disagree by much). Further, the mean trace has a definite guide to its stability, namely its variance. On the other hand, with the Monte Carlo driven mutual information measure, we essentially hope enough of the space has been explored to approximate the full integral. Thus, even if submodularity can be proven, it is not clear that the more expensive mutual information measure provides clear benefit.

5.7 Summary

This chapter has explored the placement of a single camera, then multiple cameras, along straight roads and curved roads. To be able to handle the larger search space than has been handled previously, I devised a heuristic inspired by the internal mechanism of Kalman filters. This gave some insight into the relationship between curvature and single camera placements, when optimizing for instantaneous (i.e. single-update) improvements in vehicle uncertainty. However, using it to optimize the field of view as well as orientations yielded unsatisfactory results; thus I still consider the joint optimization of field of view and orientation to be an open problem.

Using the complex metrics from the literature lead to the primary conclusion that cameras should generally be pointed along roads rather than perpendicular to them. Also, matching the diagonal of the field of view to the path does in fact provide superior results to the other discrete options I examined, but further in-depth examination is required to exclude all other possible orientations from being better. Lastly, although the metrics generally point towards the oncoming vehicle, if it is already well localized it is better to pick another high-performing orientation (e.g. along the road) a little further down the road.

Chapter 6

Conclusion

In this dissertation I have re-examined the idea of off-board cameras for vehicle localization. First, I developed a model of a camera, along with a probabilistic error bound for vehicle location predictions. Using this error model, I analysed the feasibility of the approach, showing that sub-10 centimetre positional and sub-1 degree orientational installation accuracy is required to obtain reasonable performance. Further, the algorithms calculating the centre of vehicles need to identify the corresponding pixel within a radius of about 10 pixels from the true pixel.

Next, I examine in which direction a single camera should be oriented to best reduce vehicles' localization error, which accumulates due to dead reckoning noise. A heuristic is employed, inspired by the internal mechanism of Kalman Filters, to rapidly explore the space of possible orientations and field-of-view. Results from the heuristic are then used to seed more sophisticated metrics that measure vehicle performance in simulations, in a sort of cascaded optimization. Following this I also test the placement of multiple cameras along a straight path.

A result supported throughout these last experiments is that cameras should point down the length of the road rather than face it perpendicularly. Additionally, cameras can maximise their effectiveness by viewing paths across

diagonally opposite corners of their field of view.

I also found that some metrics used in prior work, such as mutual information, are extremely computationally intensive, and do not produce significantly different results from relatively less expensive metrics, such as the mean trace of the EKF covariance.

The submodularity property that made greedy optimisation a good choice in prior work was also shown to be invalid when using mean trace of the covariance in my problem instance. The most severe violations of the property stem from camera placements with overlapping fields of view – it would be interesting to explore whether it is possible to construct sets of possible locations and orientations that are likely to be submodular most of the time (e.g. by minimizing overlap). Nevertheless, future work should consider more complex optimisation schemes, such as linear programming formulations [11].

Going forward, I have provided evidence that offboard camera-based localisation is possible and worth revisiting, and shown that if such a scheme is adopted, some orientations (away from the perpendicular) are superior for supporting autonomous robots than others. And, although I target localisation for autonomous vehicles, these ideas could be introduced to indoor localization as well.

6.1 Future Work

During the development of this work, I left several ideas partially explored. One was the use of fisheye, or other ultra-wide field of view cameras, instead of standard models. Another was a concrete implementation of a computer vision algorithm that computes the centre of a vehicle. In hindsight, the most useful next step would be to validate the feasibility of the required installation bounds, present a post-installation calibration procedure of a physical camera, and examine how performant current computer vision algorithms really are.

In the bigger picture of the research vision of autonomous urban fleets of vehicles, robots, and bicycles, a concrete validation should be followed by research on handling dynamic environments, occlusions, and tolerance of missing updates. If these problems are addressed, I believe the benefits of city-wide public localization networks will eventually see them be brought into existence.

Bibliography

[1] Lester E Dubins. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". In: *American Journal of mathematics* 79.3 (1957), pp. 497–516.

[2] Karl Johan Åström and Tore Hägglund. *PID controllers: theory, design, and tuning*. Vol. 2. Instrument society of America Research Triangle Park, NC, 1995.

[3] Margaret M Fleck. "Perspective Projection: the Wrong Imaging Model". In: *Google Scholar* (1995).

[4] Eckhard Kruse and Friedrich M Wahl. "Camera-based monitoring system for mobile robot guidance". In: *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*. Vol. 2. IEEE. 1998, pp. 1248–1253.

[5] Thomas Dall Larsen et al. "Incorporation of time delayed measurements in a discrete-time Kalman filter". In: *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. Vol. 4. IEEE. 1998, pp. 3972–3977.

[6] Bruno M Scherzinger. "Precise robust positioning with inertial/GPS RTK". In: *Proceedings of the 13th International Technical Meeting fo the Satellite Division of the Institute of Navigation (ION GPS)*. 2000, pp. 115–162.

[7] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[8] Francesco Borrelli et al. "MPC-based approach to active steering for autonomous vehicle systems". In: *International Journal of Vehicle Autonomous Systems* 3.2-4 (2005), pp. 265–291.

[9] Emanuele Menegatti et al. "Distributed vision system for robot localisation in indoor environment". In: *Proc. of the 2nd European Conference on Mobile Robots ECMR*. Vol. 5. 2005, pp. 194–199.

[10] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[11] E Horster and Rainer Lienhart. "Approximating optimal visual sensor placement". In: *Multimedia and Expo, 2006 IEEE international conference on*. IEEE. 2006, pp. 1257–1260.

[12] Eva Hörster and Rainer Lienhart. "On the optimal placement of multiple visual sensors". In: *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*. ACM. 2006, pp. 111–120.

[13] Cédric Tessier et al. "A real-time, multi-sensor architecture for fusion of delayed observations: application to vehicle localization". In: *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*. IEEE. 2006, pp. 1316–1321.

[14] Sebastian Thrun et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of field Robotics* 23.9 (2006), pp. 661–692.

[15] Robert Bodor et al. "Optimal camera placement for automated surveillance tasks". In: *Journal of Intelligent and Robotic Systems* 50.3 (2007), pp. 257–295.

[16] Mathieu Bouet and Aldri L Dos Santos. "RFID tags: Positioning principles and localization techniques". In: *Wireless Days, 2008. WD'08. 1st IFIP*. IEEE. 2008, pp. 1–5.

[17] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[18] Emanuele Goldoni et al. "Experimental analysis of RSSI-based indoor localization with IEEE 802.15. 4". In: *Wireless Conference (EW), 2010 European*. IEEE. 2010, pp. 71–77.

[19] Norman Mattern, Robin Schubert, and Gerd Wanielik. "High-accurate vehicle localization using digital maps and coherency images". In: *Intelligent Vehicles Symposium (IV), 2010 IEEE*. IEEE. 2010, pp. 462–469.

[20] Yahya Esmail Osais, Marc St-Hilaire, and R Yu Fei. "Directional sensor placement with optimal sensing range, field of view and orientation". In: *Mobile Networks and Applications* 15.2 (2010), pp. 216–225.

[21] Jing Wang, Ratan K Ghosh, and Sajal K Das. "A survey on sensor localization". In: *Journal of Control Theory and Applications* 8.1 (2010), pp. 2–11.

[22] Maximilian Beinhofer, Jörg Müller, and Wolfram Burgard. "Near-optimal landmark selection for mobile robot navigation". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 4744–4749.

[23] M Amac Guvensan and A Gokhan Yavuz. "On coverage issues in directional sensor networks: A survey". In: *Ad Hoc Networks* 9.7 (2011), pp. 1238–1255.

[24] Maximilian Beinhofer, Jörg Müller, and Wolfram Burgard. "Effective landmark placement for accurate and reliable mobile robot navigation". In: *Robotics and Autonomous Systems* 61.10 (2013), pp. 1060–1069.

[25] Maximilian Beinhofer et al. "Robust landmark selection for mobile robot navigation". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 3637–2643.

[26] Keisuke Fujii. "Extended kalman filter". In: *Refernce Manual* (2013).

[27] Aayush Bansal, Hernán Badino, and Daniel Huber. "Understanding how camera configuration and environmental conditions affect appearance-based localization". In: *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE. 2014, pp. 800–807.

[28] Maximilian Beinhofer. "Landmark Placement for Mobile Robot Navigation". In: (2014).

[29] Alberto Hata and Denis Wolf. "Road marking detection using LIDAR reflective intensity data and its application to vehicle localization". In: *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*. IEEE. 2014, pp. 584–589.

[30] Julius Ziegler et al. "Video based localization for bertha". In: *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE. 2014, pp. 1231–1238.

[31] Shunsuke Miura et al. "GPS error correction with pseudorange evaluation using three-dimensional maps". In: *IEEE Transactions on Intelligent Transportation Systems* 16.6 (2015), pp. 3104–3115.

[32] US Air Force. *GPS Accuracy*. 2016. URL: `https://www.gps.gov/systems/gps/performance/accuracy/` (visited on 05/19/2018).

[33] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[34] Liang-Chieh Chen et al. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[35] Open Source Robotics Foundation. *Car Demo*. `https://github.com/osrf/car_demo`. 2017.

[36] Transport for London. *Streetscape Guideance*. 2017.

[37] Rafael Vivacqua, Raquel Vassallo, and Felipe Martins. “A Low Cost Sensors Approach for Accurate Vehicle Localization and Autonomous Driving Application”. In: *Sensors* 17.10 (2017), p. 2359.

[38] Shih-Chieh Lin et al. “The Architectural Implications of Autonomous Driving: Constraints and Acceleration”. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM. 2018, pp. 751–766.

[39] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv preprint arXiv:1804.02767* (2018).