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 the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: js2173@cam.ac.uk

May 29, 2018

Declaration

I Joshua G. Send of Trinity Hall, being a candidate for the M.Phil in Advanced Computer Science, 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.

Total word count: TODO

Signed:

Date:

Abstract

This is the abstract. Write a summary of the whole thing. Make sure it fits in one page.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Vision

It is conceivable that, in the near future, private vehicles 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 centimeters. 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 [33]. 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 as future research avenues.

Most current localization approaches utilize global satellite navigations systems, such as GPS and its augmentations like real time kinematic (RTK) [5]. The US government reports a global 95% accuracy of 0.715 meters [27]. 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. In contrast, most authors agree autonomous vehicles can tolerate 0.2 to 0.35 meters lateral

error [32][25][15].

To achieve these error bounds, current techniques may use computationally heavy visual odometry [25], very high precision maps [15], or expensive LIDAR to localize lane markings very accurately [24].

In this work, I show that if certain installation and algorithmic constraints can be met, cameras mounted at 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 optimal camera placements and configuration for localization and navigational performance.

The results explored here could also be useful in other scenarios, such as indoor robotic localization (eg. in factories), or in other constrained environments.

1.3 Contributions

The practical questions answered by this work are the following:

1. How accurate can offboard (ie. in the environment) cameras be for localization tasks?
2. In which direction should the camera be facing and what should its properties be (for instance, field of view) to optimally aid vehicle localization and navigation, and how is related to road curvature?
3. How should a set of cameras be placed along a path to optimize navigational performance?

The first question relates to feasibility of direct camera-based localization, and is addressed at the end of Chapter 4.

The second and third questions 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. Thus, it is also closely related to previous work on surveillance networks.

While literature on surveillance using cameras has explored camera placement, few has allowed more degrees of freedom than simply adjusting the tilt (pitch) and choosing one of a limited set of positions. Further, no exploration of properties of cameras has been performed - for instance, the impact of allowing a larger field of view in the observation task. Lastly, surveillance tasks normally seek to maximize different objective functions than robotic navigational performance.

Finally, throughout this work I examine the impact of the environment, specifically road curvature, on the various tasks presented. To my knowledge, this variable has not been analysed previously.

In summary, I make the following contributions in this work

- An error model for vehicle localization using infrastructure-mounted cameras, which is motivated by physical properties such as installation uncertainty
- A feasibility analysis for using cameras for localization, using the developed error model
- An analysis of placement of a single camera camera along roads, and the relationship to road curvature
- An analysis of placement of sets of cameras along roads, and the relationship to road curvature

1.4 Overview

Chapter 2 reviews relevant literature in the fields of localization, landmark placement, and surveillance. Chapter 3 presents the simulation used for anal-

ysis, as well as background material referenced throughout this dissertation.

Chapter 4 then develops the error model for cameras used in simulation, 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 problem 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. This is hardly a new idea – sources dating back to the 90s and before 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), sonar, angle-of-arrival or other wireless signals to perform localization TODO CITE. These all suffer from the same drawbacks: measurements can be unreliable due to delayed signals or multipath effects, and plugging into these systems requires specialized hardware. Received-signal-strength techniques, which usually map the environment beforehand, then look up the current signal fingerprint, are limited to well mapped environments and suffer badly in dynamic environments. TODO CITE

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. According to civil engineering researchers at the Cambridge Department of Engineering, these approaches have generally been deemed a failure due to maintenance issues, and lack of adaptability. CITE TODO

External, vision-based monitoring systems found some support in the 90s and early 2000s. The best known work might be from Kruse and Wahl[3], 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.[8] use a slightly different approach of 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 favor, only recently appearing slightly differently in the context of surveillance tasks (see the following section).

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 extensive camera networks. Secondly, a major problem in the past was the inability to acquire the agent without visual markers on the robot. In the authors own words "... it is not easy to correctly detect the robots in dynamic environments". 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 amounts of salient information. This can include information about other objects in the scene, such as pedestrians, bicycles, etc. And, even though in this work I only consider the (x, y) world coordinates of the vehicle, images could also be used to update the heading of the robot and correct for orientational drift.

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[23], which focuses on landmark placement for mobile robots. The key difference is the use of an external sensor rather than a camera mounted on the robot. Additionally, he primarily considers circular fields of view on the robot, looking straight up, whereas I consider optimizing for position, orientation, and to an extend field of view. However, many of the formulations, including the Monte Carlo approach to evaluating mutual information (Section 5.2.2). A noteworthy contribution of this work is the use of submodularity (diminishing returns on placements) to prove good approximations to the NP-hard landmark placement problem.

Another relevant paper from the surveillance literature is [13]. Although the authors formulate a completely different objective function intended to maximize the observed length of piecewise-linear paths, their ideas inspired my computationally cheap objective function presented in Section 5.1. The authors use their objective to place a series of cameras with two degrees of freedom (location on the border of a region, and pitch). I use my metric to reduce the search space across four degrees of freedom (pitch, yaw, and vertical and horizontal field of view) into a few interesting points, which are then passed onto a more complex metric based on Beinhofer's thesis.

In [10], the authors optimize camera coverage of a map, modeling camera coverage as a 2D triangular region. They propose using binary mixed integer programming to find the global optimum of multiple camera orientations. In [21] the authors optimize directional coverage, that is the fraction of the time cameras can achieve a specific view of the object, defined by an orientation and an allowed angle about it. [16] 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. Please refer to [19] for more work on camera-based coverage models. Common to these works it their 2D treatment of the placement problem, optimizing for coverage rather than localization power, as well as using expensive linear programming-like optimizers.

Lastly, it is worth mentioning the work from Bansal, Badino, and Huber[22], who consider the problem of camera orientation on autonomous vehicles themselves. They show that the most informative region (defined by entropy) is given when the camera maximizes useful pixels – ie. focuses the most pixels on informative regions that have distinguishing characteristics, mostly along the edge of roads.

***** TODO ***** Double check to make sure no one's looked at curvature and effect on sensor placement

Maybe an image in here? It's a bit full of text

Chapter 3

Design and Implementation

This section outlines the work completed during the development of this dissertation, and provides mathematical background referenced to throughout the report.

TODO insert image of Prius model to break up the text a bit

3.1 Approach

The basis of this work is a simulation developed using industry-standard tools. The Gazebo [6] robotics simulator is used in conjunction with ROS, the Robot Operating System [14], to model a non-holonomic Ackermann drive vehicle following predefined paths. A camera is modeled using standard formulations discussed in Section 3.2.

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

The standard model for cameras is the pinhole projective model. This model keeps lines in the real world as straight lines in the pixel plane, and can be implemented as a single matrix multiplication

TODO matrices describing world =¿ pixel transformation

My implementation is a raytracing module that only uses the translation and rotation portions of the matrices above, which bring the world into place in the camera's zero-centered, Z-axis aligned coordinate system. Raytracing allows adding parametric geometry into the world, which is used later. 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 pinhol model at 110 degrees horizontally and vertically, when the perspective model breaks down [35] (some authors claim different cutoffs all the way down to 70 degrees [17], but for the analysis presented here it won't matter too much).

An infrastructure-mounted camera's task is to localize vehicles in its pixel plane into world coordinates, and provide an error bound. The error bound is developed in Chapter 4. To localize the vehicle, the camera needs to be provided with its own world orientation and location, 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.

The parameters, location, orientation and a vehicle in the pixel plane can be combined to estimate the geometric center of the vehicle, on the ground plane. Thus, the update from the camera is world location (x, y), as I assume a planar surface with $z = 0$.

In the following chapter I discuss how I model an abstract localization algorithm. A concrete implementation could use an object detection or segmentation network, such as YOLO [34] or DeepLab [29], which are constantly

improving, to determine a bounding box for the vehicle. The center of the box in the pixel plane can be seen as an estimate of the volumetric center 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 arbitrary complex algorithms, or even require vehicles carry visual markers.

3.2.1 Coordinate Systems

For reference, the following graphic shows the orientations associated with camera placements. I only allow pitch and yaw to be modified during my optimizations, as it simplifies the geometric modeling performed in following chapters.

TODO FIGURE

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 [13], where it was used to represent paths to observe in a surveillance task. This is attractive since any arbitrarily complex path can be represented the same way, with some tradeoffs in accuracy and size.

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.

TODO figure of dubins curves - original path I used?

Dubins curves can therefore be easily parametrized using only curvature and length. My implementation allows specification sequences of curvatures and lengths, which are then joined together (again using translation and rotational matrices) to form continuous curves. Dubins curves also 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

I began representation of rotations as quaternions and translations as vectors, but this was very error prone. In the end I used homogenous coordinates to jointly store translation and rotation matrices, and unit tested all geometric transformations.

In order to provide road boundaries and maintain compatibility with placement of cameras, 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.

3.4 Vehicle Model and Controls

The navigation tasks performed throughout are executed by a modified open-source model of a Toyota Prius [30]. 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 and orientation quaternion, but due to the planar

restriction 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).

TODO figure of PID feedback

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.

Typical downsides of PID are that they only control a single variable, their parameters can be difficult to tune, and that they don't perform well in out-of-ordinary scenarios. I implemented a PID controller to maintain vehicle speed, and attempted to use another for steering angle, which proved too difficult to tune well for wider variety of scenarios.

Speed Controller

The speed PID controller has access to the vehicle throttle and the 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:

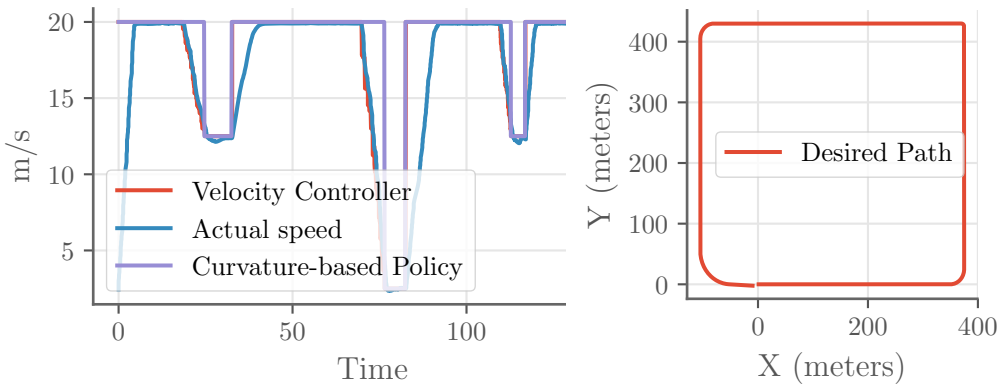


Figure 3.1: Speed Controller Performance over the calibration curve, traversed from (0,0) counter clockwise. Notice the sharp corner in the top right corner corresponds to a big drop in target and actual speed.

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

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 a 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 also only allow accelerations once the curvature decreases.

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 once constant speed but not others.

Instead, I use the steering controller published in [12], which was describes 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.

TODO image of crosstrack distance, angles

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

The original paper shows that when used with 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 [7]. Vehicle control is a complex task in itself with extensive literature, refer to [28] for a comprehensive survey. The approaches employed here were chosen mostly for their computational simplicity.

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 [20]. An EKF operates in predict and update steps, propagating the vehicle's state according to a motion model in the predict step, and cor-

recting the vehicle state during the update step if there is a measurement present. Like a standard Kalman Filter, every step produces normal distributions, which are efficient to compute over. The main difference to a standard Kalman Filter is that non-linear motion models can be used by linearizing the model at each timestep with a Taylor-like expansion about the estimated mean.

Formally, if the previous 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, 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 linearization 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 (ie. a linear relationship, which works well enough in practice). Finally, $\boldsymbol{R}$ is the covariance of the measurement.

An important 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 the 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 [9]. Here, the control commands $\boldsymbol{u}$ are not actually the commands sent to the vehicle, but taken from the "wheel odometry", which is just the simuation's ground truth state (a common formulation). So, $\boldsymbol{u}_t = (\bar{x}_t, \bar{y}_t, \bar{\theta}_t)$ is 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: the translational change, and a starting and ending angular change. These are perturbed slightly and added to the EKF's current state.

TODO figure

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

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

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

This calculates the ground truth changes to orientations and distance.

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

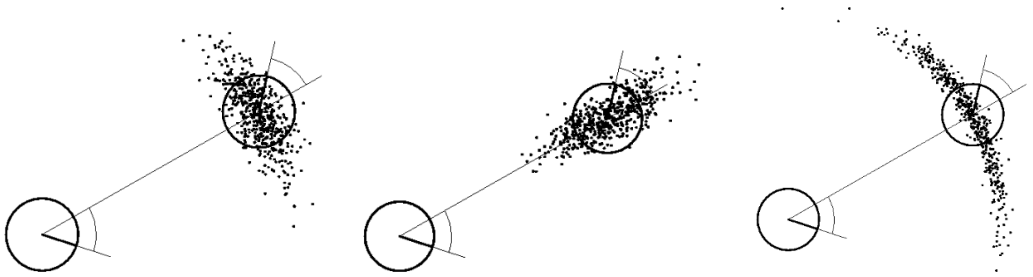


Figure 3.2: How various settings of α's affect the motion model. The canonical settings produce the leftmost spread[1]

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

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

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

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

Lastly, given current mean belief state (x_t, y_t, θ_t), we generate

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

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

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

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.

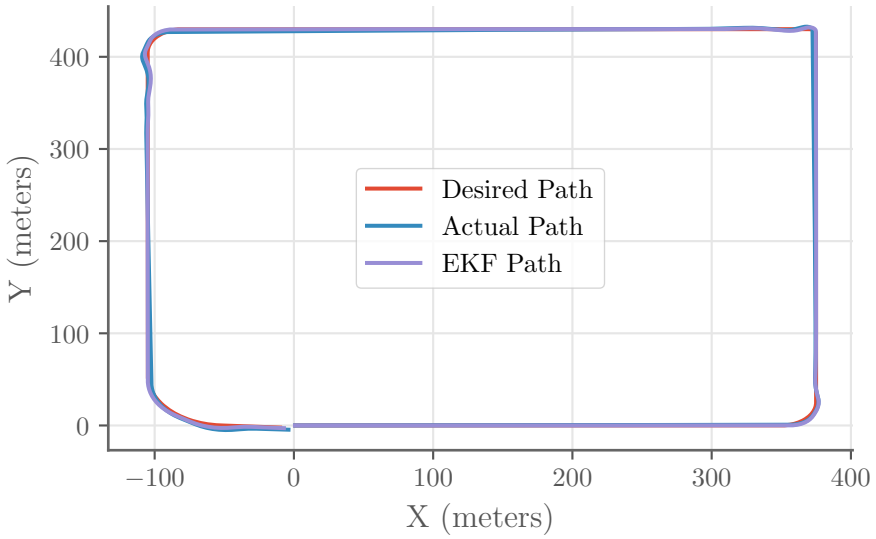


Figure 3.3: The calibration circuit containing one tight turn, two middle turns, and one low curvature turn, for a total length of about 1800 meters.

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

Lastly, since I am using a simulation without a physical reference vehicle to calibrate my parameters against, I use data from [32]. They measure their vehicle's wheel odometry over 12 laps of a circuit, 1800 meters long, similar to the one below. I mainly aim to produce similar order-of-magnitude errors.

I ran my simulation on the circuit 20 times, and compare my calibrated results to [32] 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 [32]
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 models 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) **??**, to entropy of the final robot state **??**, to the mutual information between robot states and observations **??**. Surveillance literature has also maximized the observability of a path, essentially defined as previously unseen path length in the camera's pixel plane [13].

The core difficulty of the multiple-camera placement problem is the exponential growth of possible placements. In fact, most variants of the problem are shown to be NP-hard [23].

The authors in [18] 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 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. For comparison, I also evaluate the entropy of the final robot state, and the mean value of the trace of the covariance matrix.

However, evaluating any of the objective functions even once is expensive, which is why authors have generally only optimized one or two camera settings 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 objective function which captures the localization power of a single camera placement. The best results from this metric are used as parameters for the expensive greedy optimization.

***TODO rewrite this it's ugly**

3.5.1 Entropy and Mutual Information

Two of the possible objective functions utilize entropy, which can be seen as a measure of uncertainty of a random variable. The entropy of a discrete random variable $h(X)$ is maximized 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:

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

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 5.2.2.

Chapter 4

Camera Model

The analysis presented in the Chapter 5 is based on the navigation of an autonomous vehicle guided only by dead reckoning. Errors accumulate over time need occasional updates to reduce the positional uncertainty of the robot. The sensor selected to provide this information is the camera, which is already widely deployed in modern city infrastructure.

The updates to Kalman Filters contain two components: a value (in this case, an (x, y) position), along with an uncertainty in the form of covariance. This chapter develops the simulated camera which is used to report vehicle location and uncertainty.

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 po-

sition and orientation, as well as the position and orientation of the ground plane. In 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 center of the vehicle on the ground plane. Note that in practice, using the midpoint of the front or rear axle, rather than the center, may be a more consistent reference between vehicles.

Once the computer vision algorithm running on the camera estimates the center of the vehicle, we must derive an uncertainty in the form of a covariance. I develop a mathematical model of localization error, and analyze its properties.

To start with, I develop a circular error radius, r, which I show to contain the true error 99% of the time. 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 estimated center of the vehicle
3. The ground area covered by the pixel that represents the estimated center of the vehicle (encoding camera's finite precision)

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

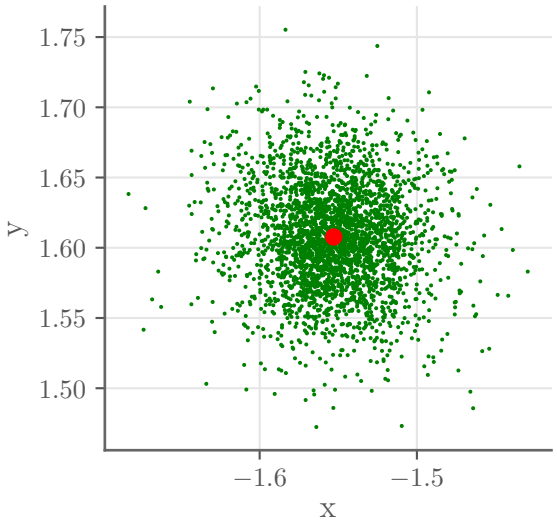


Figure 4.1: The red point is a true world location, while green points were estimated by a large sample of cameras. Discrepancies occur due to the three errors listed above. We must associate an error with each green point.

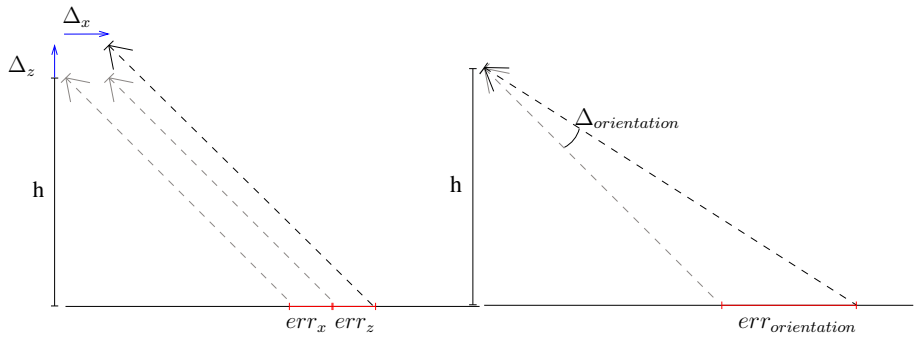
4.1.2 Location and Orientation Error

Of the three sources of error, this one is the most difficult to reason about. Many transport authorities have standard mounting specifications, including mount heights, for cameras [31]. However, none of these specify the tolerances in location and orientation, and we will have to make some educated guesses.

I assume for now that the position and orientation error is normally distributed, with 95% of positions within 0.03 meters in any direction, and orientation within a 0.5 degree cone about the principal axis 95% of the time. In mathematical terms, this means Δ_x, Δ_y, and Δ_z are all sampled from $\mathcal{N}(0, (0.03/2)^2)$. The orientation error is given as $\Delta_{orientation} \sim \mathcal{N}(0, (0.5/2)°2)$. These tight bounds could be achieved using some sort of post-installation calibration procedure.

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 $|err_{xy}| = 0.03$.



(a) Positional Errors (b) Orientational Error

Vertical translation, $\Delta_z \sim \mathcal{N}(0, (0.03/2)^2)$ 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

$$err_{xyz}(h, d) = 0.03 + \frac{0.03d}{h}$$

Similarly, it can be shown that using a conical error region about the desired principal axis of magnitude $\Delta_{orientation} \sim \mathcal{N}(0, (0.5/2)^\circ)$, for a camera at height h, and ground distance to the vehicle d, the error induced in the estimation of the vehicle's center 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 center of vehicles on the ground plane. The

center of the pixel projected onto the ground plane is taken as the vehicle center.

This process can be arbitrarily complex. 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 analyze 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 center pixel to the calculated one as $\Delta_{alg} \sim \mathcal{N}(0, \eta_{alg}/2)$.

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 center 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 installed cameras these terms have a normal distribution. The Δ_{alg} is modeled as normal as well. To take advantage of this, I also multiply each with a tuning parameter, related to their variances, which I use to trade off error bound size versus error bound accuracy. 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 center to a (slightly-mispositioned) camera, which automatically incorporates 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.

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, 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 4.3 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.

To look at how tight the error radius computation is, I present the same chart but plot the residual between predicted radius and actual error distance as a proportion of the predicted radius. Figure 4.4 shows that for the majority of predictions, the error radius is far too large – most commonly it is 3 times too large.

Note that the true error is dependent on several random variables – sampled positional and orientational error, and perturbations modeling algorithmic

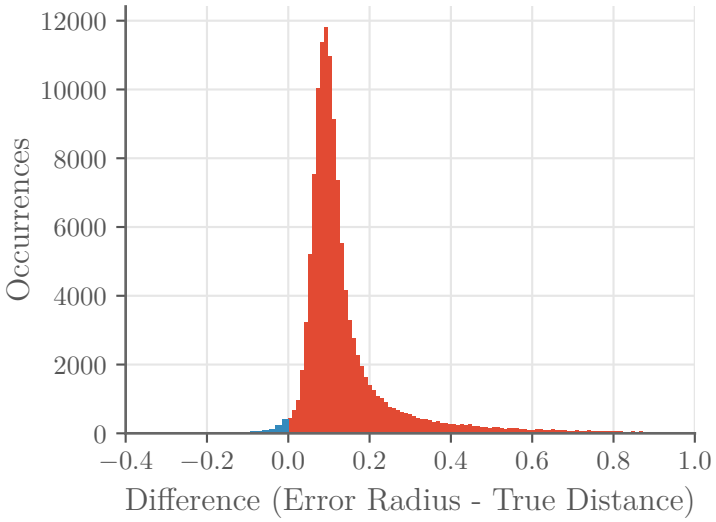


Figure 4.3: Computed radius minus actual error distance.

error (pixel ground area is not randomly sampled). Each of these are approximately Gaussian and added together r(in absolute values, meaning the computed error bound is intended to be in the tail of a rectified Gaussian distribution as well. Thus, we expect most samples to fall far from the error bound, which is exactly what we see. Attempts to make the bound tighter resulted in a loss of the 99% accuracy.

These results suggests the r is good at modeling 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 modeled 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.

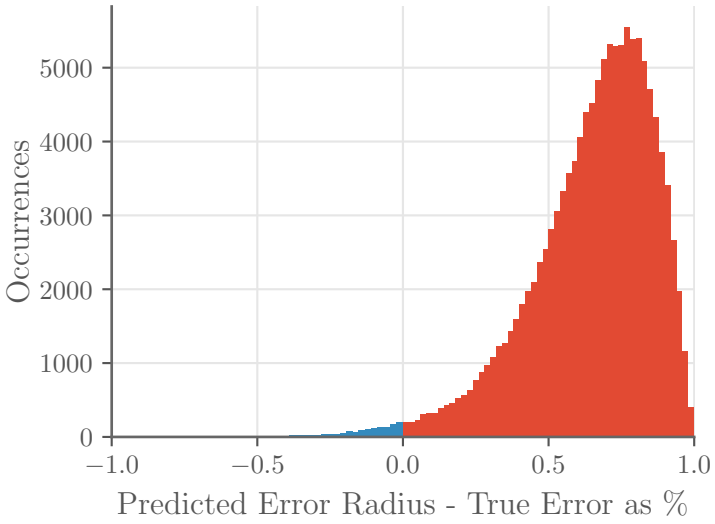


Figure 4.4: Radius error bound minus actual error distance as a proportion of radius.

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 center 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 center 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

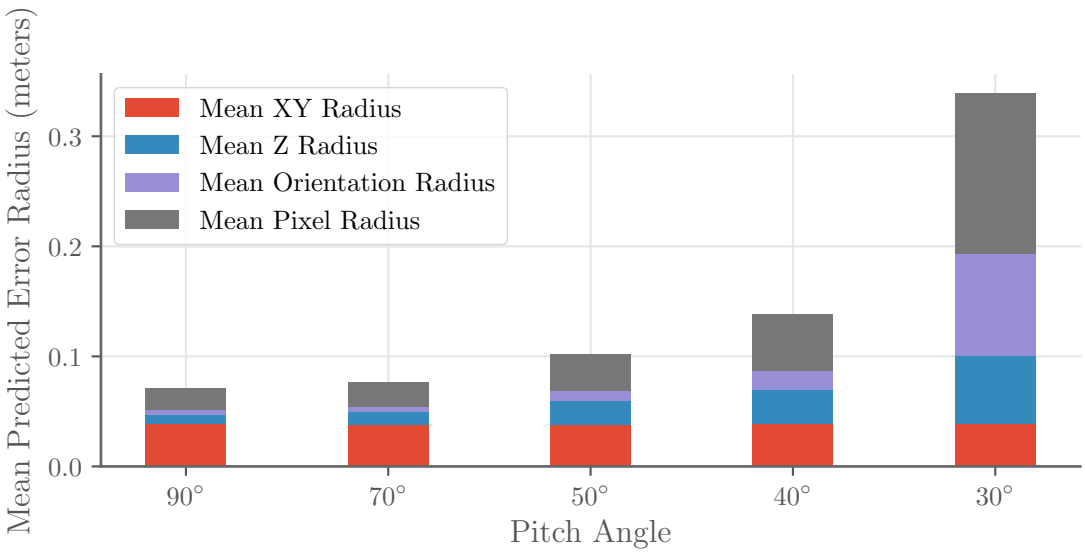


Figure 4.5: How r varies as a function pitch angle. Error bars are left out for clarity.

implies that my bound of algorithmic error is indeed well matched to the 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 questions 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

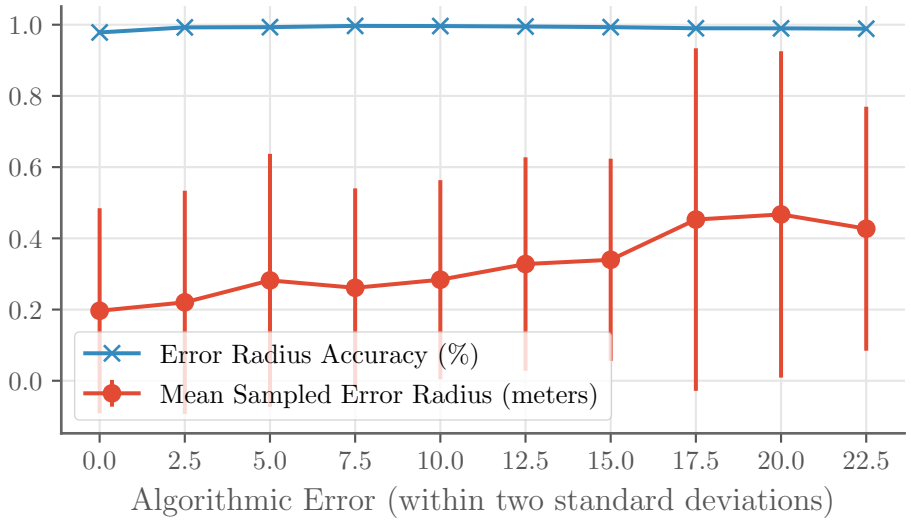


Figure 4.6: The effect of algorithmic inaccuracy on the mean value and accuracy of r. Error bars indicate two standard deviations.

further from higher resolution, which is in line with what we expect from 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 behavior 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 centimeters in any direction, orientational error within a 5° 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

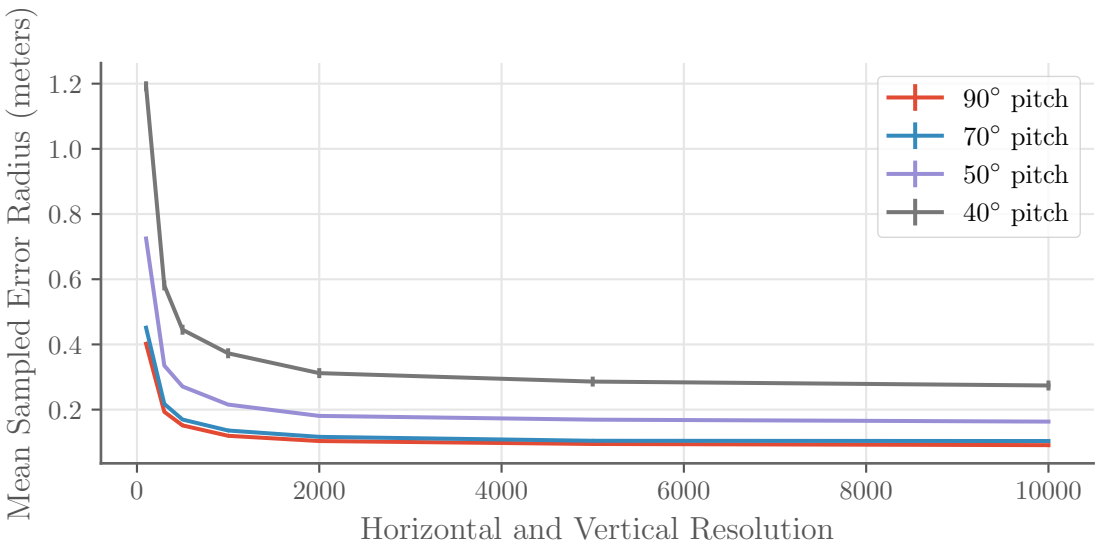


Figure 4.7: How increased resolution can increase the error bound at different pitch angles.

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. 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.

Lastly, 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-25 centimeter median localization performance.

Table 4.3: Median Localization Bound for 70° 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	**0.017**	**0.034**	**0.051**	**0.069**	**0.087**	**0.103**	**0.124**	**0.140**
0.1	**0.177**	**0.196**	**0.214**	**0.230**	**0.248**	0.268	0.280	0.298
0.2	0.342	0.360	0.378	0.394	0.409	0.430	0.443	0.460
0.2	0.503	0.524	0.538	0.552	0.566	0.586	0.605	0.616
0.4	0.665	0.685	0.702	0.715	0.737	0.757	0.765	0.794

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	**0.211**	**0.228**
0.1	**0.222**	**0.250**	0.285	0.324	0.338	0.365	0.398	0.446
0.2	0.421	0.460	0.484	0.504	0.540	0.561	0.599	0.636
0.3	0.624	0.661	0.682	0.713	0.761	0.764	0.796	0.837
0.4	0.839	0.828	0.873	0.921	0.950	0.971	1.030	1.030

These tables show that achieving a 10 centimeter installation accuracy 95% of the time is an absolute necessity, while angular deviations of up to one degree are acceptable. Finally, Table 4.5 shows how algorithmic inaccuracy impacts localization: with the usual 3 centimeter placement and a 0.5 degree orientation error as well as 45 degree pitch the algorithm must be able to localize the vehicle center to within 20 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

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.

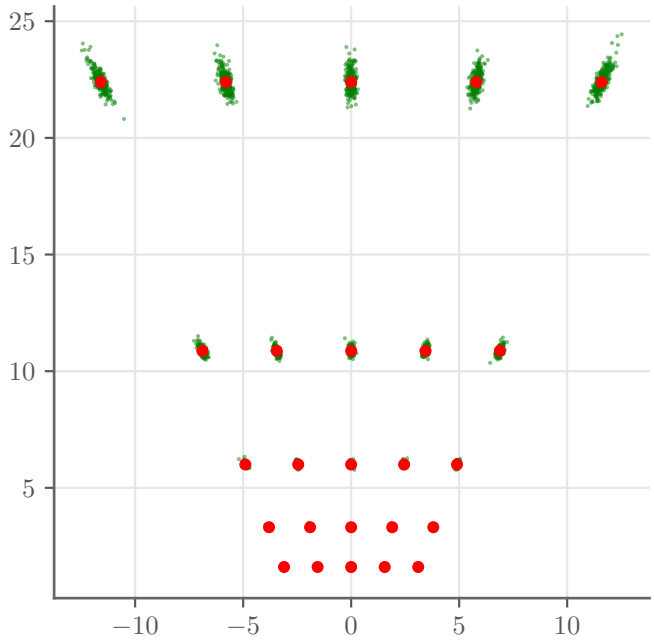


Figure 4.8: Ground plane estimates (green) of true position (red) across a large sample of cameras showing algorithmic, positional, and orientational error.

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.

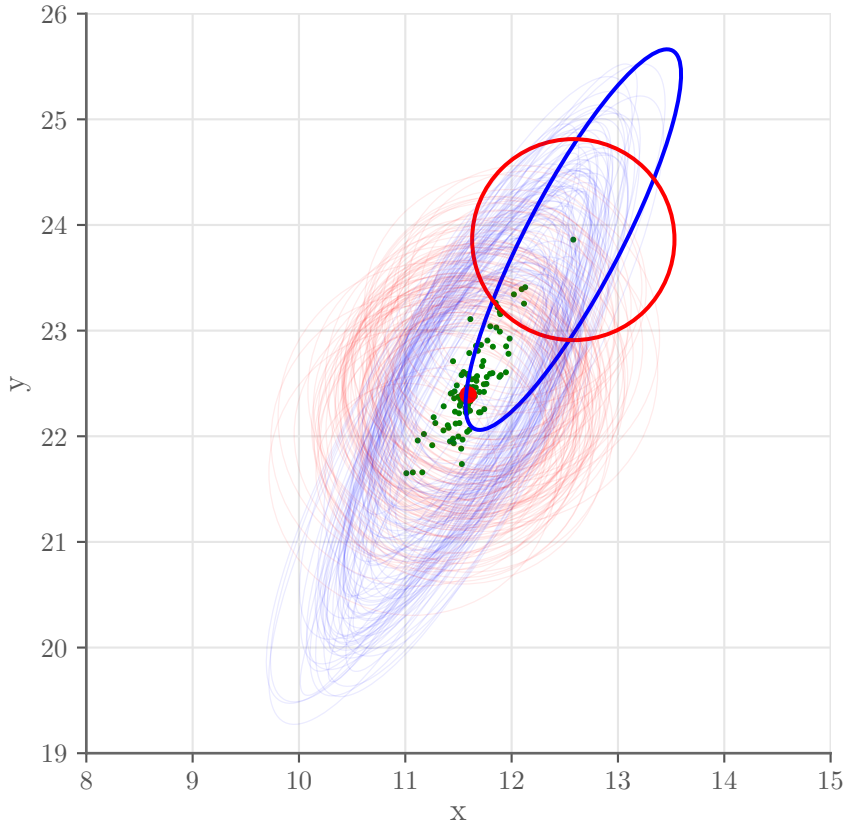


Figure 4.9: An example of where elliptical error bounds are more accurate than circular ones. Red circles represent error circles, while error ellipses are in blue. Note the vast majority of both are accurate bounds (very lightly drawn ovals and circles)

This modification improves the accuracy of the error bound even further, as shown in Table 4.6. A case of this occuring is shown in Figure 4.9 which samples many camera positions and orientations estimating the location of the point in red. The circular and elliptical error bounds that contain the red point are shown faintly in the background. One case where the circular error bound fails while the elliptical one succeeds is shown in bold.

In simulation I use oval error bounds as these more effectively represent the actual shape of errors, encoding the angle ϕ and lengths of major axis M

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**

and minor axis N into the covariance between x and y.

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

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).

To fix this, I did two things: firstly, I included a timestamp in the camera's update message, which enabled me to apply a time difference correction based on the vehicle's instantaneous velocity. This is similar to early approaches proposed for handling delayed measurements such as those in [4][11]. To further avoid partially stale updates from corrupting the vehicle's state, I also made the oval 50% more circular, as defined by the ratio of its major to minor axis lengths. Together these proved to provide adequate performance. The issue of timing delays is not considered further, though would need to be re-examined in a physical implementation.

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 center 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, camera pixel areas, and the horizontal distance to the vehicle center. 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 analyzed, 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.

I split the results in this chapter based on the objective function optimized.

5.1 A Cheap Objective Function

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 grows even faster than what previous authors in this field have dealt with. Additionally, the mutual information objective function used in the following section is an extremely expensive to compute. Due to its cost and the size of the search space, I develop a cheaper objective that can be used for optimizing the parameters of a single camera. The interesting points from this search are then used as discretized points in the full optimization.

5.1.1 Objective Function for Localization

The task of a camera is to maximally aid in vehicle localization: a natural objective is therefore to maximize the reduction in vehicle uncertainty after an update. This intuitive means we have to provide the best possible updates to the vehicles, or minimize r the error radius, in all possible updates sent to vehicles passing by the camera. I consider circular errors here, since they neatly quantify uncertainty as a single number.

To quantify "all possible updates", consider that we know roughly where vehicles will be: for vehicles to be running they must self-localize to at most $\pm 25cm$ lateral error. Additionally, if the controllers are implemented well, their position estimate should be exactly on the path to be traversed. 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 center line at all points, and call it $p(x, y)$.

Now, I can formulate the objective function as minimizing the expected reported error radius over the camera's pixel plane. In other words, for a camera Cam

$$obj_1(Cam) = \sum_{(i,j) \in Cam} p(proj_{Cam}(i,j)) * r_{Cam}(i,j) \tag{5.1}$$

where (i, j) represent the pixels in the camera plane, $proj_{Cam}$ projects a pixel onto the ground plane from the camera, and $r_{Cam}(i, j)$ is shorthand for the error radius for a prediction at pixel (i, j) as given in Section 4.1.5.

The problem with this formulation is that when allowing the camera to minimize the objective, it naturally selects views where the road is not visible at all, driving the expectation to 0. Really, the problem needs to be reformulated as a maximization to obtain sensible results. To do this, r needs to be inverted, but there are many ways to achieve this.

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.

A simple one dimensional Kalman Filter computes the gain $K = \frac{\sigma^2}{\sigma^2+\sigma^2_{measurement}}$. So, I set $\sigma = 1$ and $\sigma_{measurement} = (r/3)$, resulting in the following objective function:

$$obj_2(Cam) = \sum_{(i,j)\in Cam} \frac{p(proj_{Cam}(i,j))}{1 + (r_{Cam}(i,j)/3)^2} \tag{5.2}$$

5.1.2 Optimizing Pitch and Yaw with varying Curvature

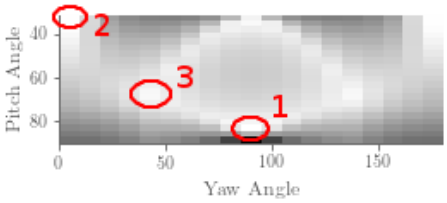
I perform an exhaustive grid search over the pitch and yaw parameters using the defined objective function. The results are presented in Figure 5.1.

The first images in a row show the evaluated objective function, the others are diagrams of the path and camera's field of view. The displayed configurations the top-scoring and distinct orientations (I ignored symmetric cases). The filled gray areas are world obstacles that I place to imitate walls.

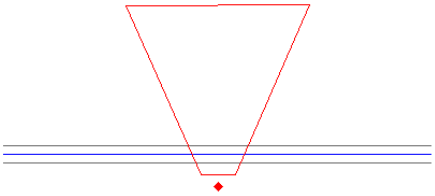
The presented optimal points match what one might intuitively expect for camera placement: ideally, we maximize both the length of the path we can spot at once, have the path trace through the highest pixel density part of the field of view – this gives the lowest error during location prediction.

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.

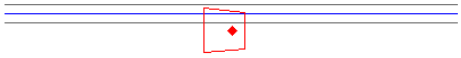
Going forward into the more expensive objectives in Section REF TODO, I use the depicted configurations as the only allowed orientations to restrict the search space as much as possible.



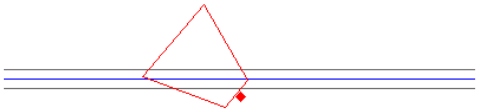
(a) 0 Curvature Scores



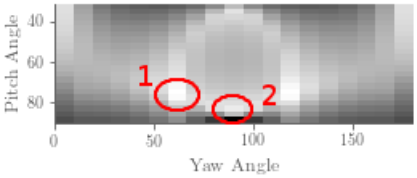
(b) Configuration #1, top score



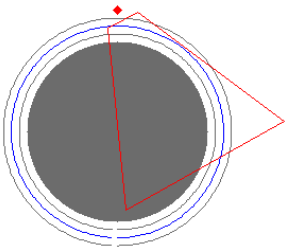
(c) Configuration #2



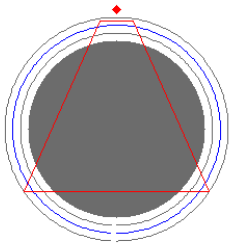
(d) Configuration #3



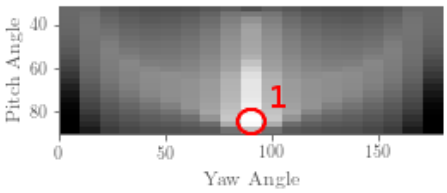
(e) 20m Radius



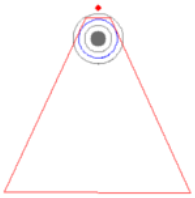
(f) Configuration #1



(g) Configuration #2



(h) 5m Radius



(i) Configuration #1

Figure 5.1: Results and high scoring orientations from the cheaper objective function.

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 [**russel2016artificial**] with random restarts to explore the space of options. The optimization takes steps in each parameter that increase the objective function, effectively zig-zagging along axes towards (local) maxima.

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

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 wide but narrow field of view. Compared to straight roads, we see more diverse solutions with high scores.

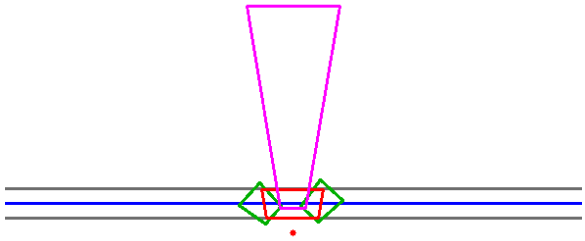


Figure 5.2: Visualization of Top Hillclimbing Result for straight roads.

Table 5.2: Top 3 results from hillclimbing across four parameters with curvature 0.05.

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

5.1.4 Discussion

The objective function used in this section can be seen as a measure of the *instantaneous localization power* of the camera. In other words, if a vehicle appeared suddenly, or could only be sent a single update, how can the camera best 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

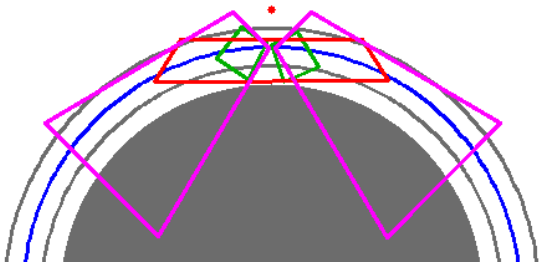


Figure 5.3: Visualization of Top Hillclimbing Result for a curvature of 0.05.

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 centered about a single path. The latter property is discussed as a future research direction at the end of this report.

Since I am essentially able to perform exhaustive searches over the possible pitch and yaw angles, I use this as a pre-optimization step to discover interesting orientations to explore further with more expensive objective functions.

5.2 Navigation-specific Metrics

Here I define alternative metrics that I use to I evaluate camera placements.

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, a measurements.

5.2.1 Entropy of the Final State

The entropy of a robot's state is a measure of the spread of its probability distribution. A peaked distribution has low entropy – all the probability is concentrated in a few regions. A flat distribution is maximally uncertain.

The robot's belief of its own state is estimated by the EKF running on board. It captures, via the mean state x_t and covariance matrix Σ_t, the distribution $p(x_t|u_{1:t}, z^A_{1:t})$.

At the end of an execution of a path, we can measure the differential entropy of this distribution as follows:

$$h(x_t|u_{1:t}, z^A_{1:t}) = \frac{1}{2} log((2\pi e)^k |\Sigma_t|) \tag{5.3}$$

Here, $|\Sigma|$ is the determinant of the covariance matrix. Since entropy is a measure of uncertainty, we try to minimize it via the camera placements.

It is tempting to try to compute the entropy of the joint robot states as the sum of the differential entropy at each time step, ie.

$$h(x_1, ...x_t|u_{1:t}, z^A_{1:t}) = \\ h(x_1|u_1, z^A_1) + ... + h(x_t|u_{1:t}, z^A_{1:t}) \tag{5.4}$$

However, this transformation would imply that the state at each time is completely independent. The actual joint entropy is the value optimized in [18], and is described in the next section.

5.2.2 Mutual Information

I use as my objective functions the one from [18], and describe it following their notation. The goal is to maximizing 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. This value is constant for a path. So, maximizing 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 $p(z^A_t|x_t)$ 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 Marlo 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 discretize using a small delta when calculating the probabilities to obtain small but finite values. However, this also requires that the full Monte Carlo approximation be run several thousand times to obtain valid results.

5.2.3 Mean Trace of Covariance Matrix

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

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

It is interesting to note that the determinant of the covariance, the trace of the covariance, and the graphical uncertainty ellipse of the EKF are all related to the eigenvalues of Σ: the ellipse's major and minor axes are the eigenvectors scaled by the square root of the eigenvalues; $tr(\Sigma)$ is the sum of the eigenvalues, and the determinant is the product of the eigenvalues. Thus, these measures can interpreted measuring the sum of the radii (squared), and the approximate area of the ellipse (squared), respectively.

5.3 Placing Single Cameras With Navigational Metrics

5.3.1 Straight Roads

5.3.2 Curved Roads

5.4 Multiple Camera Placement

* Discuss submodularity

* test submodularity by sampling – make sure minimum overlap of camera ground areas!

* straight roads + 1 curvature

* propose cascaded optimization – choose possible sensor locations, use cheaper objective to find top 1 or 2 positions or orientations, add these to the set of possible locations and orientations

5.5 Conclusion

Chapter 6

Optimizing Road Construction for Navigation

* brief - look at how to connect two points if various obstacles are put in the way * maybe try a triangular point an and rectangular block

Chapter 7

Summary and Conclusion

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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] Thomas K Sharpless, Bruno Postle, and Daniel M German. "Pannini: a new projection for rendering wide angle perspective images". In: *Proceedings of the Sixth international conference on Computational Aesthetics in Graphics, Visualization and Imaging*. Eurographics Association. 2010, pp. 9–16.

[18] 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.

[19] 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.

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

[21] Yi Wang and Guohong Cao. "Achieving full-view coverage in camera sensor networks". In: *ACM Transactions on Sensor Networks (ToSN)* 10.1 (2013), p. 3.

[22] 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.

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

[24] 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.

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

[26] 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.

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

[28] Brian Paden et al. "A survey of motion planning and control techniques for self-driving urban vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55.

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

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

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

[32] 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.

[33] 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.

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

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