A Risk Assessment Infrastructure for Powered Wheelchair Motion Commands without Full Sensor Coverage

by

Pouria TalebiFard

B.A.Sc., The University of British Columbia, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2014

Abstract

Smart powered wheelchairs offer the possibility of enhanced mobility to a large and growing population—most notably older adults—and a key feature of such a chair is collision avoidance. Sensors are required to detect nearby obstacles; however, complete sensor coverage of the immediate neighbourhood is challenging for reasons including financial, computational, aesthetic, user identity and sensor reliability. It is also desirable to predict the future motion of the wheelchair based on potential input signals; however, direct modeling and control of commercial wheelchairs is not possible because of proprietary internals and interfaces. In this thesis we design a dynamic egocentric occupancy map which maintains information about local obstacles even when they are outside the field of view of the sensor system, and we construct a neural network model of the mapping between joystick inputs and wheelchair motion. Using this map and model infrastructure, we can evaluate a variety of risk assessment metrics for collaborative control of a smart wheelchair. One such metric is demonstrated on a wheelchair with a single RGB-D camera in a doorway traversal scenario where the near edge of the doorframe is no longer visible to the camera as the chair makes its turn.

Preface

A version of this thesis is submitted for publication, in which I was the lead investigator, responsible for all major areas of concept formation, data collection and analysis, as well as manuscript composition. Junaed Sattar was involved in the latter stages of concept formation and contributed to manuscript edits. Ian M. Mitchell was the supervisory author on this project and was involved throughout the project in concept formation and manuscript composition.

This research was supported by CanWheel (the Canadian Institutes of Health Research (CIHR) Emerging Team in Wheeled Mobility for Older Adults #AMG-100925), National Science and Engineering Council of Canada (NSERC) Discovery Grant #298211, and the Canadian Foundation for Innovation (CFI) Leaders Opportunity Fund / British Columbia Knowledge Development Fund Grant #13113.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I offer my lasting gratitude to my supervisor, Ian Mitchell. I'm not convinced that words would ever suffice but, Ian: to work with you has been a real pleasure, with heaps of fun and excitement. Your high scientific standards and critical insights serves as an inspiration to me, and has provided me with a strong research foundation on which to continue building. Thank you ever so much for being so patient and encouraging.

Moreover, I feel heavily indebted to my former supervisor, Meeko Oishi, for her continued support and the opportunities she provided me with.

I am very grateful to Jim Little for his insightful comments on my thesis and motivating discussions. I have been privileged to get to know and collaborate with Junaed Sattar. I learned a lot from him about research and technical groundwork and I thank him for that. Needless to say that this work would not have been possible without Pooja Viswanathan's thesis work, which identified the problems that I set out to solve.

I also want to thank Nando de Freitas for introducing me to the field of machine learning and getting me hooked on it.

My time in graduate school was made enjoyable in large part due to friends that became a part of my life and helped me with personal challenges. I would like to thank namely Farshid Samandari for his support and for being the voice of reason in my life. I am very grateful to Chloë Filson and Robin Wilson for their friendship; and Guggy Grewal for impeding me from embarrassing myself too often.

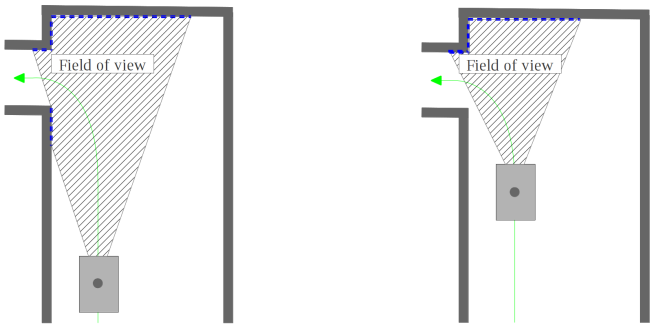
Much gratitude is owed to my family for all their love and encouragement. For my parents, Sirous and Nahid, through whose sacrifices, my siblings and I have the right to freedom of belief and the right to higher education. For my brother, Peyman, for being nothing short of the perfect role model. For my sister, Sahba, for her unconditional love and care, and for instilling love of science in me.

Chapter 1

Introduction

Mobility impairment is becoming increasingly common as the population ages, and a lack of mobility can lead to a host of further social, mental and physical impairments. Powered wheelchairs (PWCs) can greatly improve the mobility of those who are unable to self-propel in manual wheelchairs; however, existing PWC technology is often inappropriate for users with, for example, low vision, hemispatial neglect, spasticity, tremors, dementia, severe Alzheimers, or other cognitive impairment because they may not be able to safely drive these large, heavy and powerful machines [2]. Smart PWCs seek to overcome this limitation and could benefit a sizeable population [3]. Considerable progress has been made in fully autonomous PWC navigation, but there are populations for which full autonomy is undesirable; for example, older adults with cognitive impairments can become quite agitated when the PWC seems to act on its own initiative. Consequently, recent efforts have focused on smart PWCs which can constantly yet seamlessly collaborate with the user to accomplish his or her goals. In the context of this paper, the goal of the collaborative control is minimal intervention in the user's commanded motion while still avoiding collisions.

Collision avoidance requires knowledge of the surrounding environment. Because the majority of PWCs are purchased through insurance or public health care plans with significant pressures to reduce expenses, a critical factor in the success of any commercial intelligent PWC will be the cost of sensors. With manufacturer and retailer markups totalling several hundred percent on parts, few payers will be willing to cover the cost of surrounding a power wheelchair with a suite of laser rangefinders (LRFs). For cost purposes, currently available small and cheap RGB-D cameras (such as the Kinect™) are an appealing alternative to LRFs. However, cameras have much narrower fields of view (less than 60° for the Kinect) than LRFs (typically 120°–180°), and camera systems generate much more data than 2D LRFs; for example, a single Kinect nearly saturates a typical laptop's peripheral bus. Even if it were financially and/or computationally practical to completely surround a PWC with LRFs and/or cameras, it is likely that the resulting system's halo of sensors would be unappealing to potential



(a) Field of view while the adjacent wall is still visible.

(b) Field of view as PWC begins turning; adjacent wall is no longer visible.

Figure 1.1: Illustration of limited sensor coverage.

users, both for aesthetic reasons [4]—it increases the visual size of the PWC, which is already a rather bulky assistive device—and due to the potential for stigmatization [5]—it implies that the user is unable to operate a standard PWC. As a final challenge, LRFs and cameras are routinely occluded, blinded or otherwise temporarily disabled when operating in the field.

Consequently, full sensor coverage in a smart PWC system is an essentially unattainable goal, and holes in coverage can lead to collisions. As an example, a system with a front facing stereo vision camera was tested with cognitively impaired older adults in [1]. The camera's narrow field of view led to a common failure mode during the trials: The participant would drive the PWC parallel to a wall (for example, down a corridor) until it was close enough to a corner or doorway that the camera could no longer see the adjacent wall (see figure 1.1 for an illustration of this moment); if the participant then made too tight a turn the footrest or side of the PWC would hit the corner or doorframe.

In this work we describe a system which can construct a collision risk assessment for short-term trajectories of the PWC which may steer it into regions not currently visible to the sensor system. We adapt a probabilistic mapping technique [6] to construct an egocentric local occupancy map which surrounds the PWC and thereby provides information about (static) obstacles outside the sensor field of view.

In addition to the local obstacle map, a key requirement for risk assessment is accurate modeling of future trajectories. Commercial PWCs include drivetrains, power electronics, control systems and input devices from sev-

eral different companies, most of whom consider the technical specifications of and interfaces to their products to be trade secrets; thus, it is infeasible to construct models from first principles or in most cases even to interact with the control and drive system except through existing user interfaces. To work around this constraint we construct a neural net model mapping joystick motions to PWC motion.

Given a map and a method to simulate the trajectory that will be generated by a particular choice of input signal, there are many approaches to assess risk. In a fully autonomous system, this process is relatively straightforward: choose an input signal which has a low risk of collision. In contrast, in a collaboratively controlled system the user's (relatively unpredictable) input must be taken into account; therefore, the method for assessing risk must consider both the user's capabilities and the ways in which the system is willing to intervene if the risk becomes too great. Our CanWheel team is currently running Wizard of Oz trials with our target population (cognitively impaired older adults) to better understand what type of intervention will be most effective at avoiding collisions and acceptable to the users [7]. For the purposes of the experiments in this paper we propose a straw man risk assessment to demonstrate the technique. We expect that whatever scheme we design based on our trials would have to be modified for populations with different diagnoses, and therefore we focus our efforts here on mapping and modeling algorithms on top of which a variety of risk assessments can easily be constructed.

With that goal in mind, the contribution of this thesis is a system through which a measure of the short-term risk of collision with nearby objects while driving forward under collaborative control can be evaluated using only a single, low-cost, discreetly mounted RGB-D camera and without detailed knowledge of or access to the PWC control system. This task is accomplished by maintaining an egocentric probabilistic occupancy map (to overcome the camera's narrow field of view) and constructing a neural network model of PWC motion (to simulate the response of the PWC to the collaboratively chosen inputs). This map and model infrastructure can easily be modified to handle additional sensors and/or different PWCs.

We test the effectiveness of the system in detecting higher risk maneuvers when turning into a doorway from a corridor. We consider a naive risk assessment metric to demonstrate the difference in risk for a safe trajectory versus an unsafe one.

1.1 Outline

This thesis presents a system which can construct a collision risk assessment for short-term trajectories that may steer the PWC into regions not currently visible to the sensor system.

We dedicate the rest of this chapter to explore some of the more recent work done on smart wheelchairs, and the hardware and software platform that was employed for development and testing of our system.

- Chapter 2 contains the formulation and implementation details for construction of an egocentric local occupancy map. Brief discussion of relevant mapping techniques from the literature and the general mathematical formulation of the update rules for measurements and observations are also presented in chapter 2. The chapter concludes with the employed software package(s), visualization tools, and demonstration of real world results.

- Chapter 3 describes the employed model construction procedures for forward movement simulation. The chapter elucidates the data collection and necessary pre-processing steps to use the designed model. Moreover, the quality of the model after optimization of the model parameters is presented.

- Chapter 4 discusses a number of possible risk assessment schemes that could be utilized on the presented system. The chapter presents a simple measure that will be used to assess the system in chapter 5.

- Chapter 5 presents the effectiveness of the system in a scenario that replicates a common source of collision in the trials performed by [1].

The last chapter is dedicated to concluding remarks and suggestions for future research.

1.2 Related Work on Smart Wheelchairs

Several prototype intelligent wheelchair systems that attempt to compensate for lack of cognitive capacity of the user for safe locomotion and navigation have been developed over the years. Since [1, 8, 9] provide a thorough review of smart wheelchair system prototypes along with their employed range of sensors, here we focus on relatively recent work.

(a) PWC with the sensor mounted underneath the base.

(b) Close-up view of the RGB-D sensor highlighted with a white rectangle.

Figure 1.2: Photos of our PWC.

Every prototype equips PWCs with a range of sensor systems including bumper skirts [10], sonars [11–13], infrared sensors [13], LRFs [13–17], stereo vision cameras [1], and omnidirectional stereo vision cameras [18].

In all of these systems the mounting location for the sensor(s) is chosen to maximize effectiveness, rather than unobtrustive or aesthetically pleasing locations. Broad consumer and payer acceptance of smart PWC products will eventually demand a minimum of sensors placed in less than ideal locations, and our focus here is on mechanisms to evaluate risk of collision despite the degradation in sensor coverage that implies.

1.3 Hardware and Software Platform

Our PWC is a Quickie® Rhythm modified by AT Sciences to include their Drive Safe collision avoidance system [19] and custom rotary encoders on the electric motor driveshafts that we use for odometry measurement. The only component of the Drive Safe system that we use in the research described here is the CANBus to USB interface, which allows us to read the odometers and the joystick and write joystick-like commands to the PWC's motor

controller (we have removed the non-embedded components of the Drive Safe software, as well as the infrared, sonar and bump sensor pods which surrounded the PWC). Our sensor is now an Asus Xtion Pro Live RGB-D camera connected to the laptop through USB. The camera is mounted almost invisibly below the battery enclosure, looking forward from under the footrests. Figure 1.2 shows both the PWC and a close-up identifying the location of the camera.

Our PWC is equipped with a Lenovo W530 laptop with Intel® Core™ i7-3720QM processor, 8GB main memory and a 120GB solid state drive. All systems are implemented in ROS [20] (Fuerte Turtle release) on top of an Ubuntu version 12.04LTS host OS. Specific software environment and package requirements for each component of the system will be discussed in their respective chapter under the heading *implementation details*.

Chapter 2

An Egocentric Local Map

Perhaps one of the most crucial elements in developing a safer and more flexible mobile robot lies in the robot's ability to interact coherently with its surrounding, usually unstructured, environment. Intuitively, the ability to interact with the environment suggests the ability to sense, plan, and react to the stimuli inherent to the environment. For that, the robot must be able to *perceive* its surroundings using sensory information and to utilize it to plan and react accordingly. For mobile robots this first stage is often achieved with the help of maps similar to the 'map' representation in the callout in figure 2.1.

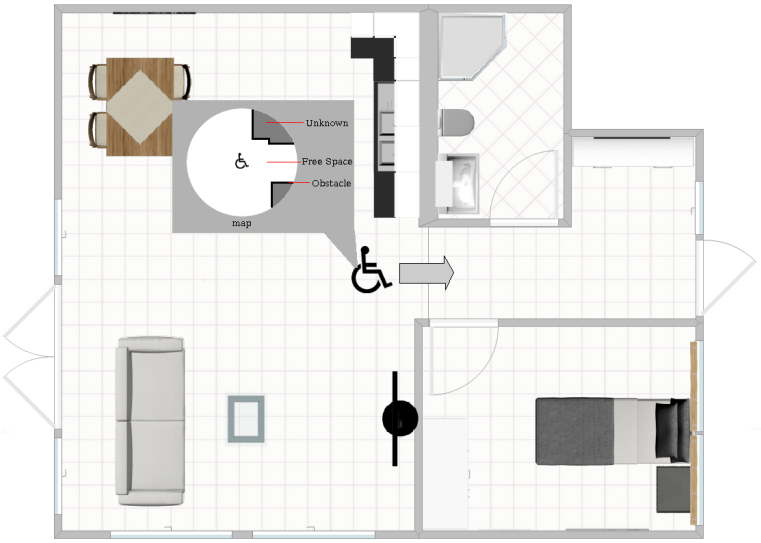


Figure 2.1: Acquiring a map representation while navigating in a cluttered environment. As illustrated in the circular 'map' representation, the dark regions correspond to the areas where obstacles reside, white regions are empty areas, and occupancy information in the dark gray areas is unknown. (*i.e.,* areas beyond the obstacles where no sensor data is available.)

In this work, we concern ourselves with the problem of acquiring the map representation for the purposes of obstacle detection and collision risk assessment. We present a probabilistic representation of an egocentric obstacle map obtained by processing range sensor data and movement readings with incremental updates using Bayesian estimation procedures.

2.1 Motivation

The aim of this work is to improve upon the free space detector in [1]. The filtering algorithm in [1] for free space detection is a simplistic procedure which only represents obstacles in the perceptual field of the sensor. Each cell in the obstacle map takes a value between 0 and 255, where 0 represents the white empty cells, as shown in figure 2.2, and 255 are the dark cells where an obstacle is believed to reside. After every sensor reading, each cell is updated by either adding a constant if an obstacle resides in the cell or subtracting a constant if the sensor reports the absence of an obstacle. This approach, although computationally cheap, responds slowly to dynamic obstacles and does not take into account the movement of the robot. For instance, the algorithm will lag in updating the map as the robot is moving towards a stationary obstacle. Another disadvantage is that the obstacle will appear in the obstacle map for a brief time after the obstacle has been removed from the perceptual field of the sensor. Moreover, since the algorithm does not make use of movement data, the algorithm will have no recollection of preceding scenes. For instance, the narrow field of view of the sensor in [1] and not incorporating odometry information led to a common failure mode during the trials: The participant would drive the PWC parallel to a wall (for example, down a corridor) until it was close enough to a corner or doorway that the camera could no longer see the adjacent wall (see figure 1.1 for an illustration of this moment); if the participant then made too tight a turn the footrest or side of the PWC would hit the corner or doorframe.

The proposed algorithm addresses the task of detection with more complex map updates relative to [1]. Specifically, the algorithm takes into account dynamic obstacles in the sensor's field of view and incorporates robot movement to update the obstacle representation.

Figure 2.2: Occupancy grid representation of the filtering algorithm presented in [1]. White cells denote empty spaces, solid black cells correspond to occupied spaces, and gray regions represent the area outside the sensor's field of view. Figure taken from [1, figure 3.5 c].

To overcome and address the shortcoming we identified from [1], we adopted an egocentric obstacle map representation in polar coordinates where the origin of the map coordinate system is always centered on the camera, as illustrated in figure 2.3. This implies that the egocentric map requires that the entire map is updated with every incoming odometry measurement, but there is no need to maintain a localization estimate within the map. While updating the map is as much as or slightly more expensive than updating a typical localization estimate, a significant computational benefit accrues during the simulation of future trajectories for risk assessment: There is no need to deal with uncertainty in the initial position from which these trajectories start. We choose the polar coordinate system because updates from the camera (or any range sensor at the origin) are cheap to apply to the map, since each reading spans along a ray in the map (as shown in figure 2.3), and movement updates are no more expensive than they would be in a rectangular coordinate system.

2.2 Related Work

Robust mapping techniques for obstacle representation and robot navigation in unstructured and unknown environments have taken different forms over the years. From edge detection methods [21] to Vector Field Histograms [22] to occupancy grid maps [6], each adopt a distinct obstacle representation

that is well suited to their more primary tasks—navigation and obstacle avoidance in most cases.

Edge-detection based obstacle detection and avoidance techniques were one of the popular methods of their time. For avoidance, the system would either steer the robot around the boundaries of the obstacle (see [21]) or take a panoramic scan and apply an edge-based global path planner to plan the subsequent steering direction(see [23]).

Occupancy grid maps are the most prominent method for spatial representation for mobile robots. They have been used in many different scenarios as local and global map representations for path planning and navigation [24, 25]. There is a large body of work [26–28] on building occupancy grid maps while estimating the location of the robot within the map.

The Vector Field Histogram (VFH) [22] method constructs a 1D histogram obstacle polar density plot to select the best steering direction for the robot. VFH is a reactive approach intended for autonomous control and relies on the presence of a localization system to navigate [29].

A notorious technique used by mobile robots for mapping is simultaneous localization and mapping (SLAM), which couples the task of mapping and localization (within that map). The depency of map estimate and pose makes the problem high-dimensional and complex [6].

2.3 Problem Formulation

In this section we establish the basic mathematical notation for our mapping algorithm.

As mentioned in section 2.1, we represent the map in a polar coordinate system; with angle $\phi \in [-\pi, +\pi]$, where $\phi = 0$ is directly in front of the PWC, and radial distance $\rho \geq 0$. The map is stored on a curvilinear grid generated by a tensor product of samples in ϕ and ρ (the white, gray and black dots in figure 2.4). The ϕ samples are equally spaced over the interval $[-\pi, +\pi]$. The ρ samples are drawn from an interval $[\rho_{\min}, \rho_{\max}]$ where $\rho_{\min}$ is chosen such that all points $\rho < \rho_{\min}$ would be within the PWC and $\rho_{\max}$ is chosen based on the maximum range of the RGB-D camera. Furthermore, the spacing of samples in the ρ direction grows quadratically, to match both the degradation of accuracy of the camera's depth readings as the distance to obstacles grows as well as the decreased importance of distant obstacles in the short-term risk assessment process.

The algorithm for maintaining the map follows a fairly standard occupancy grid mapping approach [6] with minor modifications to account for

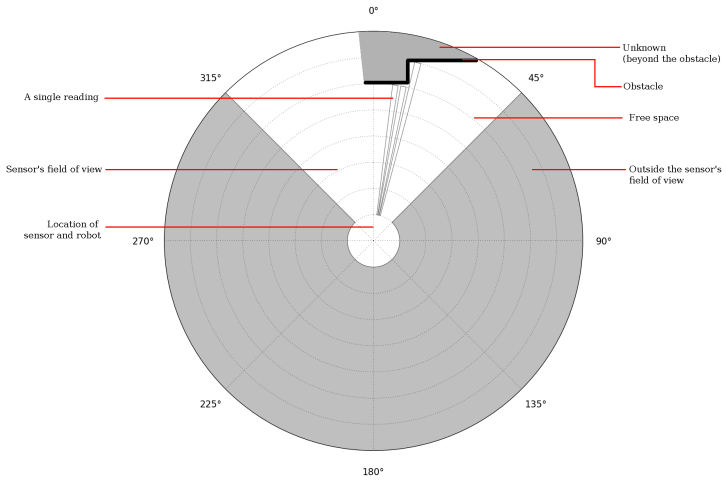


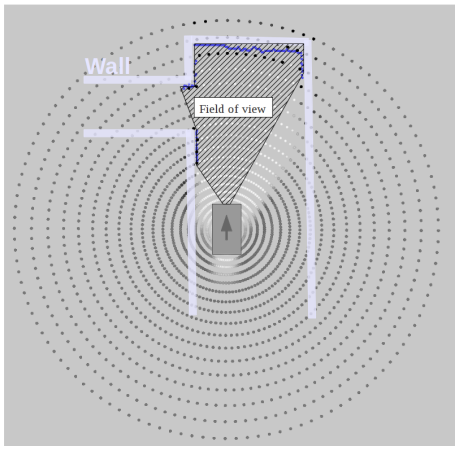
Figure 2.3: Illustration of an egocentric obstacle map representation in polar coordinates. The setup exemplifies a case where the sensor is mounted near the center of the axis of rotation. Note that only three of the range readings (rays) are displayed for illustration purposes.

limited sensor coverage and the fact that the egocentric map can change due to either obstacle or PWC movement. Our goal is to estimate the probability of a particular cell of the map being occupied at the current time given the previous maps, sensor readings, and movements. Formally, this would take the form of computing the following distribution:

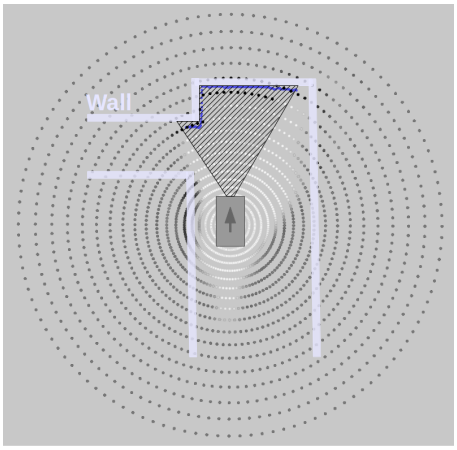
$$p(m_t(\rho, \phi)|m_{0:t-1}, z_{1:t}, u_{0:t}) \tag{2.1}$$

where $m_t(\rho, \phi)$ is the outcome that the map cell at point (ρ, ϕ) is occupied at time t, $m_{0:t-1}$ are all previous maps (for all values of (ρ, ϕ)), $z_{1:t}$ are all past observations, and $u_{0:t}$ are all odometry measurements. Assuming independence of the observations and movements allows us to factorize the probability distribution into a term dependent only on movements and a term dependent only on observations:

$$\begin{aligned} &p(m_t(\rho, \phi)|m_{0:t-1}, z_{1:t}, u_{0:t}) \\ &\quad = p(m_t(\rho, \phi)|m_{0:t-1}, z_{1:t}) \cdot p(m_t(\rho, \phi)|m_{0:t-1}, u_{0:t}) \end{aligned} \tag{2.2}$$



(a) Polar map representation before reaching the corner and while the adjacent wall is still in the perceptual field.



(b) Polar map representation where the adjacent wall is not in the field of view.

Figure 2.4: Screenshot of the polar map with actual boundaries (walls) and field of view superimposed on top. White dots correspond to empty spaces, gray dots correspond to unknown cells and darker dots correspond to occupied cells. Dark blue dots are the raw range readings.

2.4 Map Update Based on Odometry Motion Estimation

We follow the approach from [6, chapter 5] for the motion update, except that instead of updating an estimate of the PWC's localization we are updating the map. We make a Markov assumption that the new map $m_t(\rho, \phi)$ depends only on the last map m_{t-1} and the last measured odometry u_t. Regardless of the accuracy of the rotary information obtained from the encoders, the data is still prone to error as it does not distinguish slippage (or drift) from actual movements; thus, to take into account inaccuracy in the measured odometry we introduce a new random variable u'_t which is the actual odometry (if there were no slippage)

$$\begin{aligned} & p(m_t(\rho,\phi)|m_{0:t-1}, u_{0:t}) \\ & \quad = p(m_t(\rho,\phi)|m_{t-1}, u_t), \\ & \quad = \sum_{u'_t} p(m_t(\rho,\phi)|m_{t-1}, u_t, u'_t) \cdot p(u'_t|u_t) \cdot p(u_t), \\ & \quad = \sum_{u'_t} p(m_t(\rho,\phi)|m_{t-1}, u'_t) \cdot p(u'_t|u_t), \end{aligned} \tag{2.3}$$

where the final step is accomplished by observing that the measured odometry u_t is irrelevant to the map update once the actual odometry u'_t is known, and $p(u_t)$ is 1 for the measured u_t and zero otherwise. It is impractical to consider all possible u'_t, so we draw samples from a proposal distribution similar to that in [6, table 5.6], with slight modifications as outlined in Algorithm 1, where $\delta\theta$ represents the amount of rotation in the time interval $(t-1, t]$, and δx and δy represent the two-dimensional planar movement in the same time interval relative to an arbitrary external coordinate frame.

2.4.1 Movement update

As elucidated previously, the egocentric map representation requires that the robot's location remains in the center of the map; thus to incorporate movements through space, the map is shifted according to the odometry information. Given the sampled actual odometry u'_t representing the change $(\delta x, \delta y, \delta\theta)$ in the pose, one can use the relationships in (2.4) to transform the change in position from cartesian coordinates into polar coordinates, where $(\delta\rho, \delta\phi)$ represents the offset (in polar coordinates) of the origin of

Algorithm 1 `sample_motion_model_odometry`$(u_t = \{\delta x, \delta y, \delta\theta\})$:

1: $\delta_{rot} = \delta\theta$
2: $\delta_{trans} = \sqrt{\delta x^2 + \delta y^2}$

3: $\hat{\delta}_{rot} = \delta_{rot} - I_{rot}$, where $I_{rot} \sim \mathcal{N}(0, \alpha_1\delta_{rot} + \alpha_2\delta_{trans})$
4: $\hat{\delta}_{trans} = \delta_{trans} - I_{trans}$, where $I_{trans} \sim \mathcal{N}(0, \alpha_3\delta_{trans} + \alpha_4\delta_{rot})$
Given that α_1, α_2, α_3, and α_4 are robot-specific error parameters.

5: $\delta x' = \delta x + \hat{\delta}_{trans} \cdot \cos(\delta_{rot})$ where $\delta x'$ denotes the hypothesized *actual* x
6: $\delta y' = \delta y + \hat{\delta}_{trans} \cdot \sin(\delta_{rot})$
7: $\delta\theta' = \delta\theta + \hat{\delta}_{rot}$

8: **return:** $u_t' = (\delta x', \delta y', \delta\theta')$

the egocentric map due to the motion u_t'.

$$\begin{aligned} \delta\rho &= \sqrt{\delta x^2 + \delta y^2}, \\ \delta\phi &= \arctan(\delta y/\delta x) \end{aligned} \tag{2.4}$$

Given the relative motion in polar coordinates, rotations are trivial as they simply correspond to shifting elements of the angular coordinate. Translations follow the formulas in (2.5) where (ρ', ϕ') represents the corresponding offset of a general point (ρ, ϕ) in the map.

$$\begin{aligned} \rho' &= \sqrt{\rho^2 + \delta\rho^2 - 2\rho\,\delta\rho\cos(\phi - \delta\phi)}, \\ \phi' &= \arctan\left(\frac{\rho\sin(\phi) - \delta\rho\sin(\delta\phi)}{\rho\cos(\phi) - \delta\rho\cos(\delta\phi)}\right) \end{aligned} \tag{2.5}$$

Thus the update for u_t' is formally given by

$$p(m_t(\rho, \phi)|m_{t-1}, u_t') = p(m_{t-1}(\rho', \phi')|m_{0:t-2}, z_{1:t-1}, u_{0:t-1}) \tag{2.6}$$

where $p(m_{t-1}(\rho', \phi')|m_{0:t-2}, z_{1:t-1}, u_{0:t-1})$ is taken from the egocentric map at time $t-1$ sampled on the grid.

Implementation Details

The algorithm was initially simulated in MATLAB on a proof of concept basis and then implemented in PYTHON and Robotic Operating System

(ROS) for the robotic wheelchair. We discuss a few issues that arose during implementation in the remainder of this section.

Interpolation of the right hand side of (2.6) is required as it is highly unlikely for (ρ', ϕ') to be a grid node. In the MATLAB implementation, linear interpolation proved too dissipative, while piecewise cubic splines created ringing artifacts near steep discontinuities in the probability functions. In our experience, the "cubic" scheme [30] in MATLAB worked well. In the case of two-dimensional interpolation, the cubic scheme fits a bicubic surface using the values of the (sixteen) closest points.

For the PYTHON implementation, we used the Clough-Tocher piecewise cubic interpolant [31] (as implemented in the `scipy` library). The Clough-Tocher piecewise cubic interpolant is a curvature-minimizing interpolant. In our experiments, among the options available for 2D interpolation, the piecewise cubic provided a reasonable tradeoff between efficiency, accuracy and smoothness.

Because the map's grid is finite, it is also necessary to provide boundary conditions to evaluate (ρ', ϕ') that lie outside the grid. Periodic boundary conditions are the clear choice for the ϕ direction. In the ρ direction we set the probability of obstacles to zero for $\rho' < \rho_{\min}$, under the assumption that the hole in the center of the map is entirely occupied by the PWC and hence cannot contain obstacles. For $\rho' > \rho_{\max}$ we use a constant prior probability of occupancy p_{occ} (the same prior as is used in the observation model below).

We repeat (2.6) for each node (ρ, ϕ) in the grid to compute $p(m_t|m_{t-1}, u'_t)$ for a single sample u'_t. To compute the sum in (2.3) we sample a small number (eg: $j = 5$) of u'_t, and assume equal $p(u'_t|u_t)$ for all of them (eg: $1/j = 0.2$).

Algorithm 2 describes the steps involved in updating the map based on the relative odometry information. In practice, it would be a waste of computational power to account for motions that are less than the resolution of the map; hence, the function is only executed when the cumulative translations and/or rotations since the last update are above a preset threshold (derived from the user specified map resolution).

2.5 Updating the Map based on the Observation

We adapt a standard occupancy grid mapping algorithm based on a binary Bayes filter [6, sections 4.2 and 9.2] to an egocentric map. Distribution $p(m_t(\rho, \phi)|m_{0:t-1}, z_{1:t})$ in (2.2) can be computed recursively by applying

Algorithm 2 `odometry_motion_update`$(m_{t-1},\ u_t = \{\delta x, \delta y, \delta\theta\})$:

if δtranslations $>$ threshold$_{tran}$ $\|$ δrotations $>$ threshold$_{rot}$ **then**
  map$_{tmp} \leftarrow$ initialize J copies of map m_{t-1}
  for each sample $j = 1 : J$ **do**
    for all map grid nodes **do**
      $[u_t'^{(j)}] =$ `sample_motion_model_odometry`(u_t) (Algorithm 1)
      $\delta\rho^{(j)}, \delta\phi^{(j)}) \leftarrow (\delta x'^{(j)}, \delta y'^{(j)}, \delta\theta'^{(j)})$ (2.4)
      $(\rho'^{(j)},\ \phi'^{(j)}) \leftarrow$ compute the coordinates in the old system (2.5)
      map$^{(j)}_{tmp} \leftarrow$ interpolate occupancy information based on $(\rho',\ \phi')$
    end for
  end for
  $map = \frac{1}{J}\sum_{j=1}^{J}$ map$^{(j)}_{tmp}$
end if

Bayes' rule. Assuming that each observation is conditionally independent from previous observations and maps given the current map, we can write

$$p(m_t(\rho,\phi)|m_{0:t-1}, z_{1:t}) = \frac{p(m_t(\rho,\phi)|z_t)p(z_t)p(m_t(\rho,\phi)|m_{0:t-1}, z_{1:t-1})}{p(m_t(\rho,\phi))p(z_t|m_{t-1})}. \tag{2.7}$$

In the remainder of this derivation, let $m_t = m_t(\rho,\phi)$ for notational simplicity. The standard binary Bayes filter assumes static state, but we wish to allow for moving obstacles so we assume a dynamic dependence between the current and past maps of the form

$$\begin{aligned} &p(m_t|m_{0:t-1}, z_{1:t-1}) \\ &\quad = p(m_t|m_{t-1})p(m_{t-1}|m_{0:t-2}, z_{1:t-1}), \end{aligned} \tag{2.8}$$

where $p(m_t|m_{t-1}) = p_{\text{trans}}$ is a constant giving the probability that an occupied map location at the last time remains occupied at the current time. We then transform into odds form by dividing $p(m_t|m_{0:t-1}, z_{1:t})$ through by its complement $1 - p(m_t|m_{0:t-1}, z_{1:t})$, and take the logarithm for mathematical convenience. This transform conveniently cancels a few terms in (2.7) and, after incorporating (2.8), yields a simple and mostly additive recursive update

$$\begin{aligned} l_t = &\log\left(\frac{p(m_t|z_t)}{1-p(m_t|z_t)}\right) - \log\left(\frac{p(m_t)}{1-p(m_t)}\right) \\ &+ \log\left(\frac{p(m_t|m_{t-1})}{1-p(m_t|m_{t-1})}\right) + l_{t-1}, \end{aligned} \tag{2.9}$$

where we define the current log-odds as

$$l_t = \log\left(\frac{p(m_t|m_{0:t-1}, z_{1:t})}{1 - p(m_t|m_{0:t-1}, z_{1:t})}\right)$$

and set the initial log-odds to be

$$l_0 = \log\left(\frac{p(m_0)}{1 - p(m_0)}\right) = \log\left(\frac{p_{\text{occ}}}{1 - p_{\text{occ}}}\right). \tag{2.10}$$

The terms on the right hand side of (2.9) are based on, respectively, knowledge gained from the current sensor reading $p(m_t|z_t)$ (commonly referred to as the *sensor model*), the prior probability of occupancy $p(m_t) = p_{\text{occ}}$, the prior probability of occupancy remaining static $p(m_t|m_{t-1}) = p_{\text{trans}}$, and the previous log-odds l_{t-1}. The prior probability of occupancy p_{occ} is set to a value between 0.2 and 0.5, depending on obstacle density [32]. The p_{trans} is the transition probability of occupancy from $t-1$ to t. In our experiments, a value between 0.9 to 0.95 seemed to worked well. Note that the priors p_{occ} and p_{trans} are assumed to be constant (in both time and space).

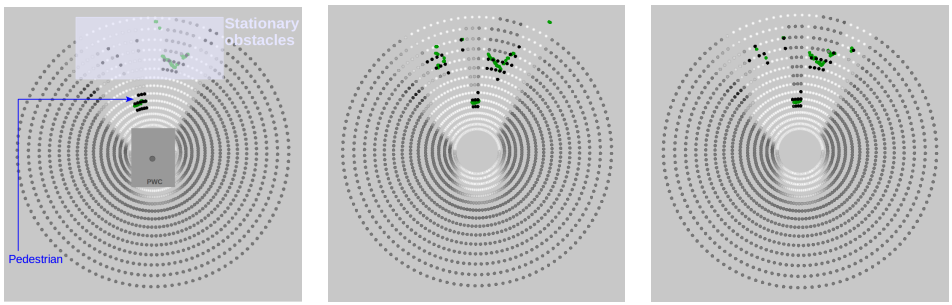
For the sensor reading term $p(m_t|z_t)$ we assume that the measurement z_t provides data in the form of the range to the nearest obstacle for a large number of angles (compared with the resolution of our grid in the ϕ coordinate) within the field of view. We assume independence between columns of the grid (all elements with the same ϕ)—not an ideal assumption, although the oversampling of the sensor offsets the issue somewhat. To perform the update on a column of the map, consider a fixed $\phi = \hat{\phi}$. In order to reduce the effect of noise in the sensor data, the range estimate for this column $r(\hat{\phi})$ is the median among all sensor range estimates coming from angles whose nearest neighbor among the columns of the map is $\hat{\phi}$. The sensor reading term is then given by

$$p(m_t(\rho, \hat{\phi})|z_t) = \begin{cases} \mathcal{N}(r(\hat{\phi}), \sigma(r(\hat{\phi}))), & \text{for } \rho \leq \hat{\rho}; \\ p_{\text{occ}} & \text{otherwise;} \end{cases} \tag{2.11}$$

where $\mathcal{N}(r, \sigma)$ is a normal distribution with mean r and standard deviation σ, $\sigma(r(\hat{\phi}))$ is chosen according to experimental measurement of the sensor's noise (an increasing function with range for the Kinect, as given in [33]), $\hat{\rho}$ is the first sample of ρ in the grid such that $\hat{\rho} > r(\hat{\phi})$, and $\rho > \hat{\rho}$ is the unobserved region beyond the sensed obstacle. In [33] the standard deviation varies based on where in the 2D image frame the data is coming from and the depth measurement being reported. In our implementations

we do not take into account variations across the 2D image frame. However, we do incorporate the variation in standard deviations with respect to depth, which increases quadratically with distance.

To account for moving obstacles, cells beyond the obstacles in the field of view are driven towards the unknown likelihood, which is l_0 in (2.10). The update rule is a simple exponential decay; where the time constant is chosen based on the rate at which sensory information is coming in. Effects of this update are shown in figure 2.5. The scenario depicted in figure 2.5 is where a pedestrian moves across the field of view from left to right and pauses momentarily in the middle of the field of view. As a result of the simple update (*i.e.,* the exponential decay), cells that are situated beyond the obstacles are forced towards to l_0, as illustrated in 2.5a and 2.5b. Meanwhile, the cells where the pedestrian occupied in 2.5a are driven towards low likelihood (or free) in 2.5b.



(a) Pedestrian enters the field of view on left and walks across. Cells beyond the pedestrian are initially white but are being driven towards gray (unknown).

(b) Pedestrian pauses in front of the sensor in the middle of the field of view. Cells beyond the pedestrian are driven towards unknown. The region where the pedestrian was situated in the previous frame is now marked free.

(c) Pedestrian stays in front of the sensor in the middle of the field of view. Cells beyond the pedestrian are turning dark gray.

Figure 2.5: Performance of the algorithm to dynamic obstacles. Occupancy information for the gray circles (cells) is unknown, white cells are free and dark (black) cells are occupied. Green points are the sensor readings. The scenario elucidates the response of the algorithm to dynamic obstacles such as pedestrians. In this scenario, the pedestrian enters the field of view from left and walks across with a momentarily pauses half way through. The cells behind the PWC are white because the the chair was moved slightly forward after initializing the map.

Implementation Details

Map grid parameters such as the radial and angular resolution can to be user-defined. For instance, it is likely that a reading with a narrow standard deviation σ will not be noticed if the abscissa representing the radial coordinate (ρ) do not intersect the normal distribution in (2.11) near its peak. To overcome the scenario where the radial coordinate is coarsely represented, we use the first-order finite difference of the normal cumulative distribution (cdf) such that

$$p(m_t(\rho, \hat{\phi})|z_t) = \begin{cases} \mathcal{C}(r(\hat{\phi}) + h(\rho)) - \mathcal{C}(r(\hat{\phi})), & \text{for } \rho \leq \hat{\rho}; \\ p_{\text{occ}} & \text{otherwise;} \end{cases}$$

where h is the width of abscissa in ρ and $\mathcal{C}(r)$ is the normal cumulative distribution function at r. Similar to (2.11) the range estimate for each column $r(\hat{\phi})$ is the median among all sensor range estimates coming from angles whose nearest neighbor among the columns of the map is $\hat{\phi}$.

2.6 Software Platform

The model developed in the preceding sections was initially implemented in a simulator and then ported over to the hardware platform introduced in section 1.3.

The simulation environment we used to initially test the algorithm was **A**utonomous **MO**bile **R**obots **sim**ulator (AMORsim) [34], designed for development and testing of localization, mapping, path planning and obstacle avoidance algorithms in MATLAB. Figure 2.6 is a screen-shot of the simulator. The range sensor in the simulator was modified so that it emulates the same noise and limitations as the RGB-D sensor that was to be employed on the PWC.

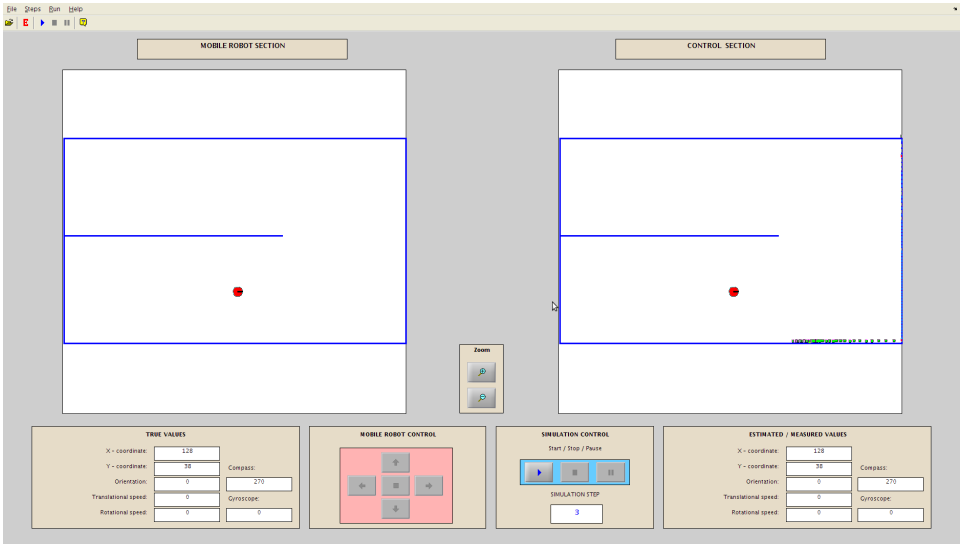


Figure 2.6: Screen-shot of the AMORsim simulator.

After development and testing of the algorithm in the simulator environment, the algorithm was ported over to Robot Operating System (ROS) [20] as a PYTHON node. In order to extract range data from the RGB-D sensor,

we used the `pointcloud_to_laserscan` package in ROS that converts 3D Point Cloud into a 2D laser scan. The movement update and sensor update routines make use of ROS's *spinning* procedure, where each routine runs on a separate thread but the map (which is stored in the form of a row-major array) is protected via semaphores. We used PYTHON's *threading* module for locking.

2.7 Visualization

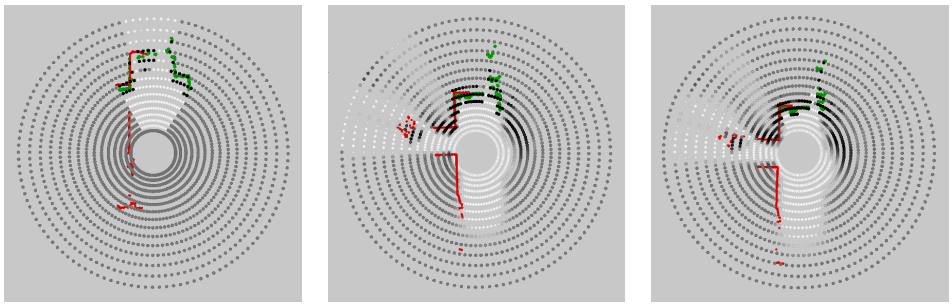
In order to visualize map updates in AMORsim, we used MATLAB's `surf` plots, which proved very helpful in debugging.

Visualization in ROS was not as seamless. Due to the polar coordinate system, we were not able to use any of the built-in visualization routines for mapping. Instead, the maps are represented as Point Clouds where cells with higher likelihood of being occupied are darker in colour and empty cells are lighter. By default, gray points represent unknown or not yet observed regions, as shown in figure 2.7.

2.8 Real World Results

In this section we present the performance of the developed algorithm. Response of the algorithm to dynamic obstacles was presented in section 2.5. Results of the movement update are illustrated in figure 2.7. The setup consists of the RGB-D sensor facing in the forward direction and a SOKUIKI UTM-30LX scanning LRF mounted to the side such that there is some overlap between the field of view of the two sensors. Note that the LRF readings are used merely for error evaluation, not map updates.

The scenario in figure 2.7 depicts the situation where the PWC is driven down the corridor shown in figure 1.1. The PWC makes a 90° counter clockwise in-place turn once at the doorway, pauses momentarily and makes a 90° clockwise in-place turn back (as shown in figure 2.7b) and proceeds moving down the corridor (see figure 2.7c).The readings of the SOKUIKI sensor are displayed in red on the map and confirm the accuracy of the movement update. Further evaluation of the scheme is undertaken in chapter 5



(a) Initial frame without any movement updates. Cells in the field of view of the RGB-D sensor are updated according to the sensor update scheme described in section 2.5

(b) The PWC moves forward and makes a 90° counter clockwise in-place turn once at the doorway. After a short pause the PWC makes a 90° clockwise turn back. The obstacles that are no longer in the field of view of the RGB-D sensor have shifted according to the movement information.

(c) The PWC moves slightly forward again. The wall beside the opening on the left is now completely outside the field of view of the RGB-D sensor. The occupancy information reported here after the movement update is consistent with the readings of the SOKUIKI sensor overlaid on top of the map.

Figure 2.7: Illustration of updating the map using odometry information. Green points are the RGB-D sensor readings and red points are the measurements from a SOKUIKI scanning range finder mounted facing sideways with some overlap with the RGB-D sensor's field of view. Dark circles (cells) correspond to occupied cells, white cells correspond to empty cells and occupancy information in the gray region is unknown.

Chapter 3

Non-Linear Control Function Estimation

High-level modeling tools are useful in the development and simulation of locomotion strategies, wayfinding algorithms and devising risk-assessment criteria, to name a few. In our application, this corresponds to modeling the dynamics of motion of a commercial wheelchair. Access to a reliable model that can predict the behavior of the robot in the immediate future is vital to the generation, optimization and control of *safe* movements.

As discussed previously in chapter 1, reverse engineering the control scheme and system identification is infeasible for commercially available PWCs. The motion dynamics of PWCs can not easily be deduced using naive Newtonian mechanics. Anybody who has driven a PWC knows that they have both significant inertia and a small but nontrivial delay when responding to joystick commands.

The motivation for this work is the "black box" problem. The black box can be described as a system whose transfer function models the desired input-output relations. Applications of the black box system ranges from control systems [35] to recognizing hand-written digits [36] and predicting energy usage in buildings [37]. Perhaps a tangible example is Suddarth's *et al.* [38] work, who utilized neural networks to control the thrust of a lunar lander using only the lander's altitude, fuel and velocity as inputs while it descends to the moon's surface.

We present a multi-layer feedforward neural network architecture for predicting the position of (the wheels of) a differential drive mobile robot in the immediate future (*i.e.,* position at $t + 1$) using eight input signals:

(i) current joystick position v_t,

(ii) change in joystick position since last sample $\delta v_t = v_t - v_{t-1}$ (to account for control system delay),

(iii) wheel rotation since last sample u_t (a proxy for PWC velocity), and

(iv) change in wheel rotation since last sample $\delta u_t = u_t - u_{t-1}$ (a proxy for PWC acceleration),

where each joystick input has two components (forward/backward for linear motion and left/right for angular motion) and each wheel rotation input has two components (left and right wheel). The output of our model is estimates of the left and right wheel rotation at the next sample time $u_{t+1} = f(v_t, \delta v_t, u_t, \delta u_t)$. Given all eight inputs at time zero $(v_0, \delta v_0, u_0, \delta u_0)$ and the joystick positions at future times v_t for $t = 1, 2, \ldots$, the model can then be used to predict the wheel rotations at future times by feeding back the outputs and appropriate finite differences to the inputs at each step; for example,

$$\begin{aligned} u_1 &= f(v_0, \delta v_0, u_0, \delta u_0) \\ u_2 &= f(v_1, (v_1 - v_0), u_1, (u_1 - u_0)) \\ &\vdots \end{aligned}$$

The process of modeling such behavior can be characterized as solving for a direct (or forward) kinematic function [39]. Earlier attempts to model the PWC's response to joystick inputs with standard linear, piecewise linear and linear autoregressive-moving-average (ARMA) models did not produce sufficiently accurate results, so we turned our attention to nonlinear models.

The motivating factor for employing neural networks is due to their ability to approximate an arbitrary mapping of one vector space (inputs) to another vector space (outputs) [40]. The notion that neural networks are universal approximators was demonstrated more than two decades ago in [41–43]. In particular, the claim surrounds multi-layer preceptrons with a single intermediate layer. The proposition suggests that with enough hidden units, a single layer neural network can approximate any function defined on a compact domain [44]; hence the use of a multi-layer feedforward neural network with one hidden layer in this work. While there are advantages in using neural networks for such applications, it should be noted that there are drawbacks that might not qualify them for all applications of this nature. It is not trivial to retrieve or extract structured knowledge from the network (*i.e.* they are "non-parametric"). Moreover, should the circumstances of the problem change, such as changes in the number of input parameters or their biasing, then the network needs to be completely re-trained.

In the remainder of this chapter we describe the data collection and pre-processing necessary to use the model we designed, the model and its construction procedure, and the quality of the results. It should be noted that we have used this procedure to construct two different models of the

PWC for two different drive system parameter settings; the quality of the model was similar in both cases.

3.1 Data Collection and Preprocessing

The empirical data used for training and validation consists of roughly 60 minutes of joystick commands v and the resulting odometry measurements u. All measurements were timestamped by ROS when they arrived from the CANBus to USB device driver. Roughly half of that driving time was on tiled surfaces and half on carpets. After temporal resampling of u and v (described below) and computing the δu and δv inputs, the data points were randomly permuted to remove any potential bias due to temporal ordering.

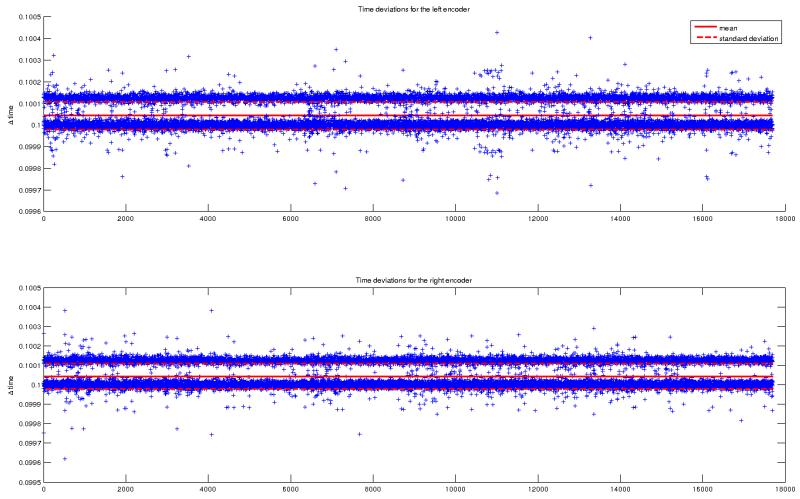


Figure 3.1: Time deviations between samples for the left encoder (top) and right encoder (bottom).

In the discrete time modeling paradigm that we adopted, all inputs must be delivered simultaneously. Unfortunately, the PWC platform delivers encoder and joystick data at different rates. Encoder data arrives at 100ms intervals, with much less than 1% timing jitter (see figure 3.1). Note that although two distinct bands appear in time deviations for both the left and

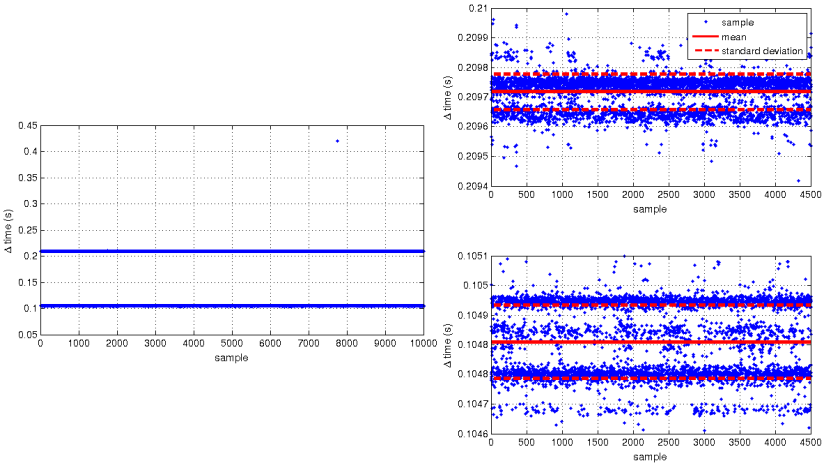


Figure 3.2: Time deviations between samples for the joystick commands. The distribution is bimodal with a large spread. Note that the subplot to the left is the overall view of all samples' time deviation. The top right subplot is the zoomed view of the time deviations of the top band (near 210ms) and the bottom right subplot is the zoomed view of the samples near the band at 105ms.

right encoder, we treat samples to be in a single band at the median since the jitter in time is minimal. The time between joystick data samples is either around 105ms or 210ms (roughly equal chance of either), again with less than 1% jitter from these two possibilities, as shown in figure 3.2. Similar to the encoder data, although there are multiple bands in the zoomed views (see top right and bottom right subplots of 3.2), notice that the jitter in time is less than 1% regardless. For the discrete time model, we arbitrarily decided to synchronize the input sequence with a fixed 200ms sampling interval.

For the training data, linear integration and interpolation are used to resample u and v into a time series with fixed 200ms period. Run-time resampling is performed whenever a joystick sample v_t arrives. Linear interpolation between joystick samples is used to estimate the effective v_{t-1} (joystick position 200ms earlier) and thereby approximate δv_t (as shown in figure 3.3). Quadratic interpolation through the previous three encoder samples is used to estimate u_{t-1} and (by extrapolation) u_t, and thereby δu_t

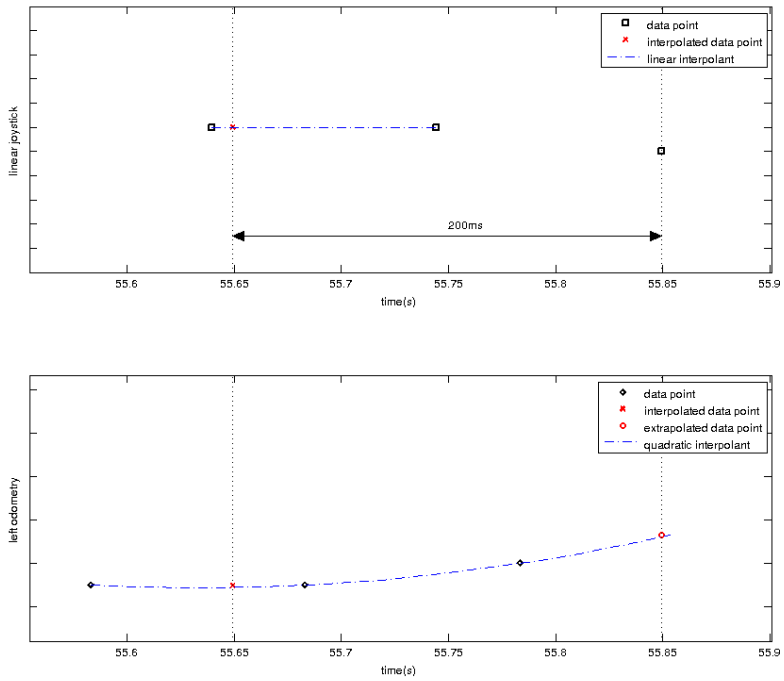


Figure 3.3: Run-time resampling of the joystick (top) and one of the encoders (bottom). To estimate v_{t-1} linear interpolation is used. Quadratic interpolation through the previous three encoder samples is used to estimate u_{t-1} and (by extrapolation) u_t.

(see figure 3.3). More specifically, since deviations of the encoder data are minimal from the 10Hz frequency, we used an interpolant with fixed abscissa that will be inexpensive to evaluate. Namely, we used a Lagrange basis set of the form

$$B_j(x) = \prod_{i=0, i\neq j}^{n} \frac{(x - x_i)}{x_j - x_i} \tag{3.1}$$

where the three bases B_j can be precomputed ahead of time and stored for run-time evaluation. Note that using linear interpolation during resampling corresponds to assuming that the PWC's acceleration or the joystick's velocity is constant over the interpolation interval, which is not an unreasonable

assumption over a 200ms time span.

The next stage in preprocessing the data is normalization. Data normalization, or any linear scaling operation that corrects for large dynamic ranges in one or more dimensions, is a prominent step in input-output setup for neural networks. It reduces the effect of large variations dominating subtle trends in the data and reduces the likelihood of underflows and overflows. Hence, the input joystick commands were remapped between $[-1, 1]$ since the minimum and maximum ranges are known. Encoder data was normalized to a zero mean and unit standard deviation to preserve outliers, such that

$$\hat{x} = \frac{x - \bar{x}}{\sigma_x}$$

where $\hat{x}$ is the standardized data, $\bar{x}$ is the mean of the original data of x and σ_x is the standard deviation of x. Based on our experience, normalization speeds up the training phase (*i.e.,* faster convergence), and prevents inputs with large scales from biasing the solution.

3.1.1 Data Sets

Of the 60 minutes of collected data, 30 minutes labeled as the *training set* was used for inner loop training (*e.g.,*: weight adjustments in the backpropagation approach), 15 minutes labeled as the *optimization set* was used in the outer loop training to optimize network parameters, and 15 minutes labeled as the *validation set* was reserved for validation. Note that only the samples in the training and optimization sets were randomly permuted. Samples in the validation set are from a continuous interval and are kept in their temporal ordering.

3.2 Neural Network Training

A multi-layer feedforward neural network is characterized by an input layer, an output layer and any number of intermediate layers, conventionally referred to as hidden layer(s).

Training of a neural network using the supervised learning paradigm can be thought of as modeling a function $f(x)$ that maps input $X = \{x_n\}_{n=1}^{N}$ to target values $Y = \{y_n\}_{n=1}^{N}$, such that $f(x_n; \theta) \approx y_i$. Consider a neural network with one hidden layer, consisting of J hidden units and I input units; then the output of the k^{th} neuron (out of K output units) would be

of the form:

$$f_k(x) = \sum_{j=1}^{J} \theta_{jk} a_j(x) + b_k$$

where θ_{jk} is the weight of the connection between hidden unit j to output unit k, and b_k is the bias on the output layer. The $a_j(\mathbf{x})$ function is usually a bounded, non-linear and differentiable function, referred to as the *activation function*. In this work we make use of the hyperbolic tangent function as the activation function.

$$a_j(x) = \tanh(\sum_{i=1}^{I} \theta_{ij} x_i + b_j)$$

where θ_{ij} is the weight of the connection between input unit i and hidden unit j, and b_j is the bias on the hidden layer. Discussions surrounding the choice of activation function are beyond the scope of this thesis and will not be addressed; nevertheless, it is noteworthy to mention that using a linear activation function would severely restrict the functionality of the hidden units and the resulting network would not differ from a network with direct connections from input layer to output layer.

Regardless of the learning paradigm that the network is trained in, the objective of the training step is to determine the weights and biases that minimize a cost function.

The frequentist and Bayesian approaches were tested for training the neural network. The conventional frequentist approach believes that there is a single set of weights that map between the inputs and outputs, whereas the Bayesian approach believes a probability distribution over the weights is appropriate.

3.2.1 The Frequentist Approach

The frequentist learning approach is based on adjusting the weights of the network according to a learning policy, given a training set of input-output pairs. The procedure of the training step is a gradient-based optimization process where the objective is to minimize some measure of error or cost function. Conventionally, the cost function is defined as the mean square error (MSE) to quantify a measure of difference between the model's output, f, and the target value,y. Total error, E, over all N training samples would be of the form:

$$E(\theta) = {}^1\!/_2 \sum_i \sum_k (f_k(x) - y_k)^2 \tag{3.2}$$

The approach of using derivatives of the error function with respect to the weights was introduced by Rumelhart, Hinton, and Williams in [45]. In order to improve the convergence speed of gradient descent, a scaled conjugate gradient search strategy [46] was employed for adjusting the weights.

Detailed derivation of the rules for steepest descent on the error surface have been thoroughly described in [45]; thus, it would be redundant to repeat them here. Briefly, the update rule of the gradient descent method is:

$$\theta \leftarrow \theta - \eta \nabla E(\theta)$$

where η is the *learning rate*. Generally, η is tuned to reduce the effect of overfitting and underfitting. Finding a suitable learning rate is however not trivial. A common practice to identify an appropriate learning rate is cross validation [47], where η is optimized based on the performance of the model using an "optimization" set, different from that of the training set.

Scaled Conjugate Gradients

Gradient descent using backpropagation does not necessarily produce the fastest convergence [48] since it uses one fixed step size. Conjugate gradient methods [49] use conjugate directions instead of the local gradient information. Formally, each learning iteration, conjugate gradient methods use line search to find η that minimizes E for fixed values of θ and ∇E. Møller proposed the scaled conjugate gradient (SCG) [46], which would scale the step size based on error reduction and quadratic approximation of the likelihood to avoid performing a line-search at each learning iteration. Specifically, η at iteration n is calculated as,

$$\eta(n) = 2(\eta(n) - P^T H(n) P + \eta(n)\frac{\|P\|^2}{\|P\|})^2$$

where H is the Hessian of the gradient and P is the conjugate system. SCG has been one of the most widely used iterative methods for solving linear equations [50] and provides significant speed ups. In fact, it outperforms conjugate gradients and quasi-Newton algorithms for small gradient evaluations [51].

Prediction

Lastly, the prediction step using the frequentist approach is simply evaluation of $f_k(x^{(n+1)}; \theta_{optimum})$, where $\theta_{optimum}$ is a single set of optimum weights that minimize the cost function in (3.2).

Note that the network parameters—the number of hidden units, the coefficient of the weight-decay prior ζ and the number of epochs—were determined through an outer Bayesian optimization loop, which will be discussed in section 3.3.

3.2.2 The Bayesian Approach

The Bayesian framework can be distinguished by its ability to 1) incorporate prior knowledge to propose a prior probability distribution for model parameters, and 2) express future observations in terms of a probability distribution over all unknown quantities. Suppose the likelihood of our feedforward neural network is: $p(y_i|\theta, x_i)$, where y_i is the target of the network defined by $f_k(x_i, \theta)$. And suppose the prior distribution of the weights, θ, are $p(\theta)$. Then, via Bayesian inference, the posterior distribution of the weights are:

$$\begin{aligned} p(\theta|x_{1:n}, y_{1:n}) &= \frac{p(y_{1:n}|\theta, x_{1:n})p(\theta|x_{1:n})}{p(y_{1:n}|x_{1:n})} \\ &= \frac{p(y_{1:n}|\theta, x_{1:n})p(\theta)}{\int p(y_{1:n}|x_{1:n}, \theta)p(\theta)d(\theta)} \end{aligned} \tag{3.3}$$

for n data points. Predicting the target value y_{n+1} given the new input(s) x_{n+1} would be of the form,

$$\begin{aligned} &p(y_{n+1}|x_{n+1},(x_1, y_1), \ldots, (x_n, y_n)) \\ &\qquad = \int p(y_{n+1}|x_{n+1}, \theta)p(\theta|(x_1, y_1), \ldots, (x_n, y_n))d\theta \end{aligned} \tag{3.4}$$

The high-dimensional integral in (3.4) is not analytically tractable and is computationally complex to approximate for large number of parameters [52].

In the interest of computational tractability, work in [53] and [54] proposed using Gaussian approximations for the posterior distribution of the network parameters (weights and biases). Such approximation techniques, however, fail to provide satisfactory results in training complex models [44]. Moreover, assuming the form of the posterior distribution may not be a valid assumption in all applications.

Standard Metropolis-type methods are a more flexible approach as they sample from the posterior distribution of the weights. Using Metropolis algorithm [55] we can approximate (3.4) using S samples as

$$p(y_{n+1}|x_{n+1}, (x_1, y_1), \ldots, (x_n, y_n)) \approx \frac{1}{S}\sum_{t=1}^{S} p(y_{n+1}|x_{n+1}, \theta_t) \tag{3.5}$$

where as the limit S increases the approximation converges to the true value and $p(\theta|(x_1, y_1), \ldots, (x_n, y_n))$ is the stationary distribution of the algorithm. This however is not feasible for complex networks and is generally very slow [44]. Hamiltonian (or Hybrid) Monte Carlo (HMC) [56] method is a form of Metropolis algorithm. The advantage of using an adaptive technique like HMC is that instead of random proposal for moving particle in the standard Metropolis, it uses Hamiltonian equations to propose new positions.

In this work we make use of an adaptive technique similar to the work in [57]. We employ HMC for sampling from the weight's posterior distribution and Bayesian regression for the prediction step. Network parameters and HMC's hyperparameters were determined through an outer Bayesian optimization loop with Expected Improvement (EI) as the acquisition function [58]. Results and discussions surrounding the outer loop Bayesian optimization will be covered in section 3.3.

Hamiltonian Monte Carlo

The disadvantage of using the standard Metropolis algorithm to evaluate the integral in (3.4) is the random walk. In high-dimensional weight space, generating proposal states by randomly perturbing all weights is problematic as it is very likely that the proposed direction of motion will be rejected (by the Metropolis algorithm). In contrast, the HMC algorithm makes use of the gradient information (from backpropagation network) to drive changes in weights in directions that will have a high probability of being accepted.

Formally, HMC drives proposals similar to a physical system. It models the motion of the samples via Hamiltonian dynamics of motion:

$$\mathcal{H}(\mathbf{q}, \mathbf{p}) = \mathcal{U}(\mathbf{q}) + \mathcal{K}(\mathbf{p})$$

where $\mathcal{U}(\mathbf{q})$ and $\mathcal{K}(\mathbf{p})$ are the potential energy and kinetic energy terms, respectively. The potential energy term can be observed by:

$$\mathcal{U}(\mathbf{q}) = -\log p(\mathbf{q}) - \log(z) \tag{3.6}$$

where $p(\mathbf{q})$ is the distribution we want to sample from and z is any constant. Note that $\mathbf{q}$ is the "position" vector (as in Metropolis algorithm) and $\mathbf{p}$ is the "momentum" vector. In case of our network, $\mathbf{q}$ corresponds to the set of network weights θ. Thus, (3.6) is simply

$$\mathcal{U}(\theta) = -\log p(\theta|(x_1, y_1), \ldots, (x_n, y_n)) - \log(z)$$

Exploring in HMC takes the form of a deterministic move and a stochastic move. The former is to explore regions over with $\mathcal{H}$ is constant and the

latter is to explore regions with different values of $\mathcal{H}$. The deterministic moves follow Hamiltonian equations for motion where the derivatives of $\mathbf{q}$ and $\mathbf{p}$ are computed with respect to a "time" variable τ such that

$$\frac{d\mathbf{q}}{d\tau} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}} = \mathbf{p} \qquad \text{and} \qquad \frac{d\mathbf{p}}{d\tau} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} = -\nabla \mathcal{U}(\mathbf{q})$$

In practice, however, Hamiltonian dynamics can merely be approximated by discretization in finite time steps. *Leapfrog* [59] is the most commonly used method for this approximation [44]. In [56] stochastic movements are drawn from (3.7) though any Metropolis move that changes $\mathcal{H}$ is permissible.

$$p(\mathcal{K}(\mathbf{p})) = (2\pi)^{\frac{D}{2}} \exp(-\mathcal{K}(\mathbf{p})) \tag{3.7}$$

where $\mathcal{K}(\mathbf{p}) = 1/2|\mathbf{p}|^2$ and D is the dimension of θ.

Given the dynamics of motion, HMC can be summarized as below for S number of samples.

1. Randomly pick a direction λ for the trajectory to simulate; where $\lambda = +1$ (forward trajectory) and $\lambda = -1$ (backward trajectory) are equally likely.

2. Leapfrog: given the current $(\mathbf{p}_t, \mathbf{q}_t)$ take L steps with ϵ step size as:

$$\begin{aligned}
\mathbf{p}(\tau + \frac{\epsilon}{2}) &= \mathbf{p}(\tau) - \lambda \frac{\epsilon}{2} \nabla \mathcal{U}(\mathbf{q}(\tau)) \\
\mathbf{q}(\tau + \epsilon) &= \mathbf{q}(\tau) - \epsilon \mathbf{p}(\tau + \frac{\epsilon}{2}) \\
\mathbf{p}(\tau + \epsilon) &= \mathbf{p}(\tau + \frac{\epsilon}{2}) - \lambda \frac{\epsilon}{2} \nabla \mathcal{U}(\mathbf{q}(\tau + \epsilon))
\end{aligned}$$

   resulting in candidate $(\hat{\mathbf{p}}_{\mathbf{t+1}}, \hat{\mathbf{q}}_{\mathbf{t+1}})$

3. Accept or reject $(\hat{\mathbf{p}}_{\mathbf{t+1}}, \hat{\mathbf{q}}_{\mathbf{t+1}})$, as in Metropolis algorithm, by

$$(\hat{\mathbf{p}}_{\mathbf{t+1}}, \hat{\mathbf{q}}_{\mathbf{t+1}}) = \begin{cases} (\hat{\mathbf{p}}_{\mathbf{t+1}}, \hat{\mathbf{q}}_{\mathbf{t+1}}), & \text{if } U < \mathcal{H}(\hat{\mathbf{p}}_{\mathbf{t+1}}, \mathbf{q}_{\mathbf{t}}) - \mathcal{H}(\mathbf{p}_{\mathbf{t}}, \mathbf{q}_{\mathbf{t}}) \\ (\hat{\mathbf{p}}_{\mathbf{t+1}}, \hat{\mathbf{q}}_{\mathbf{t+1}}) & \text{otherwise;} \end{cases}$$

   for U sampled uniformly from $[0, 1)$.

The number of steps L is the number of iterations to arrive at the candidate $(\hat{\mathbf{p}}_{\mathbf{t+1}}, \hat{\mathbf{q}}_{\mathbf{t+1}})$. Small values for L will impede the algorithm from exploring and setting too large of a value for L will increase the number of samples that will be rejected and will reduce the number of stochastic moves. Aside from the network parameters, number of samples S, number of steps L and step size ϵ are optimized by the outer-loop optimization scheme as presented in section 3.3.2.

3.3 Outer-Loop Optimization

The goal of an optimizer is to find the solution that minimizes some objective function; however, this task can become computationally expensive depending on the complexity of the function and number of dimensions [60].

3.3.1 Bayesian Bandit Optimization

Bayesian optimization is an elegant framework for optimizing cost that does not require gradient information about the function and is well suited for expensive functions [57].

Bayesian optimization can be characterized by its use of 1) a kernel matrix for the Gaussian process prior and 2) an acquisition function for determining a policy of exploitation and exploration of the error surface. For the purposes of this project, we used the automatic relevance determination (ARD) Matèrn $5/2$ kernel, which is a widely used kernel for practical optimization problems [58]. The acquisition function can be described as the policy that decides the next point that should be evaluated, with a trade off between exploitation and exploration. The acquisition function that we employed in this optimization process is the Expected Improvement, which picks the minimum between the next function evaluation and the current *best* function value.

3.3.2 Optimization Results

The measure of performance that we took into account for optimization is the MSE of the output neurons from the target values. MSE was computed on the optimization set, different from the training and validation sets.

For the SCG method, we optimized the model based on the number of hidden neurons, coefficient of weight decay prior (ζ), and the epochs (the number of iterations over the data set in order to train the network). Table 3.1 lists these parameters along with the ranges over which they were set to optimize and the returned optimal values.

For the Bayesian framework, we optimized over the number of hidden neurons, number of samples to be drawn, number of steps (L), step size (ϵ), and coefficient of weight decay prior (ζ). Likewise, table 3.2 lists the parameters, their ranges, and the returned optimal values for the probabilistic model.

Parameters	**Range**	**Optimal value**
Number of hidden units	[4, 20]	19
Coefficient of weight-decay prior	[0, 1]	0.0125
Number of epochs	[1, 1000]	781

Table 3.1: Parameters of the network trained via the scaled conjugate gradient approach, the ranges over which Bayesian optimization was performed along with the achieved optimal values after 15 iterations.

Parameters	**Range**	**Optimal value**
Number of hidden units	[4, 20]	17
Coefficient of weight-decay prior	[0, 1]	0.3375
Number of samples	[100, 1000]	1000
Number of steps := L	[10, 100]	91
Step size := ϵ	[0, 1]	0.0011

Table 3.2: Parameters of the network trained via Hamiltonian Monte Carlo, their ranges over which Bayesian optimization was performed along with the achieved optimal values after 68 iterations.

3.4 Model Validation and Performance

The models' performance metric is the MSE. The overall MSE over the optimization data set (see section 3.1.1) for the network trained by SCG and HMC are 0.03695 and 0.05124, respectively, which are based on running the Bayesian optimization algorithm over 15 iterations on the frequentist model and 68 iterations on the probabilistic model. The performance of the models on the validation data set and run-time extrapolation scheme discussed in 3.1—are comparable to the performance of the models on the resampled data. The MSE over the validation data set for the network trained by SCG and HMC are 0.03887 and 0.05029, respectively.

A comparison of the predictive capability of the models with respect to the actual target values is plotted in figures 3.4 and 3.5 for 20 random time samples from the validation set. Notice that the predicted outputs in figure 3.4 contain no indication of uncertainty whereas predicted outputs in figure 3.4 are bounded by error bars. The error bars are the result of Bayesian approach to prediction that it does not use a single *best* set of network weights.

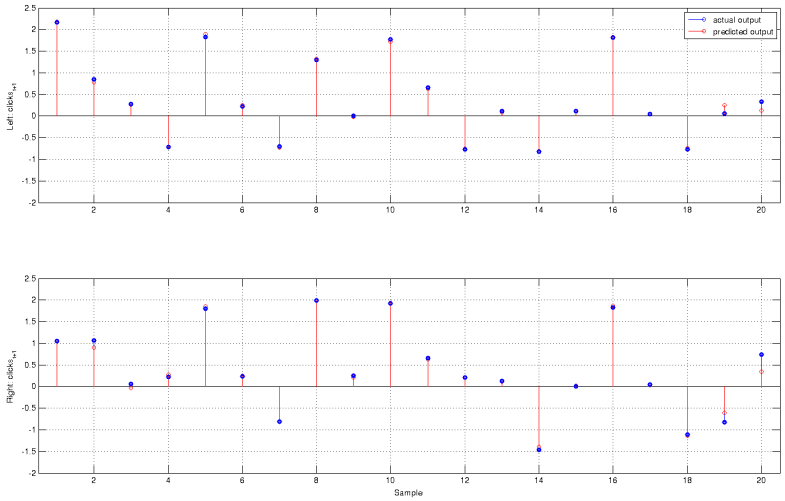


Figure 3.4: Predictive capability of the model trained via SCG.

Plots of the relative error of the models' output with respect to the actual target values are shown in figures 3.6 and 3.7. The resulting errors were unbiased (centered at zero). The mean error magnitude is less than 15% for both the deterministic and probabilistic model.

The minimum MSE over the number of iterations for the Expected Improvement and a random grid search procedure are illustrated in figure 3.8. Optimizing the parameters of the deterministic model (see: 3.8, top) via EI as the acquisition function proved to outperform the randomized grid search. Likewise, EI outperformed Randomized search greatly in optimizing model parameters for the probabilistic model (see: 3.8, bottom).

The performance of the probabilistic approach (via HMC) proved similar to the deterministic (via SCG) backpropagation (likely because the problem is not particularly high dimensional); thus, for the purposes of this work we used the simpler deterministic model.

3.5 Implementation Details

The process for training the model via SCG and HMC takes about 4 hours and 1.6 hours, respectively, and on an Intel® Core™ i5-660 CPU with 4GB

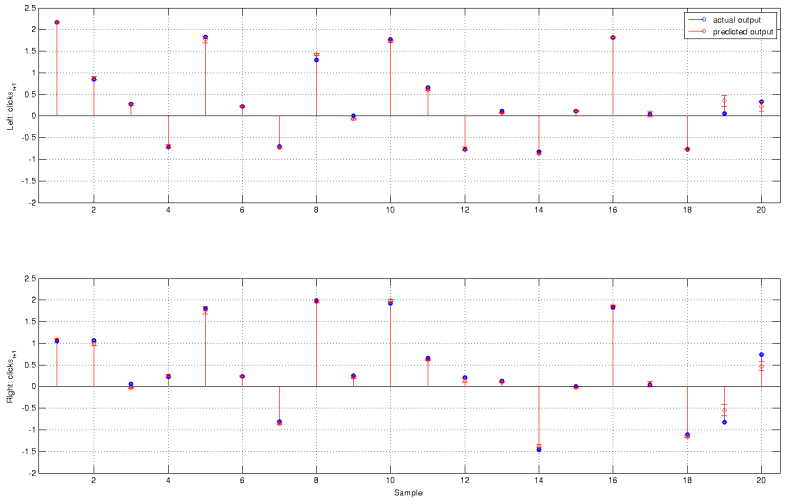


Figure 3.5: Predictive capability of the model trained via HMC algorithm.

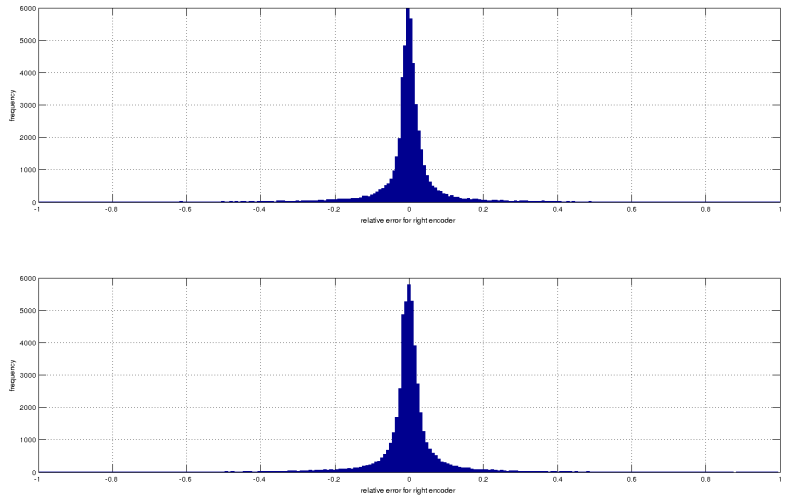


Figure 3.6: Distribution of relative error of the model trained via SCG.

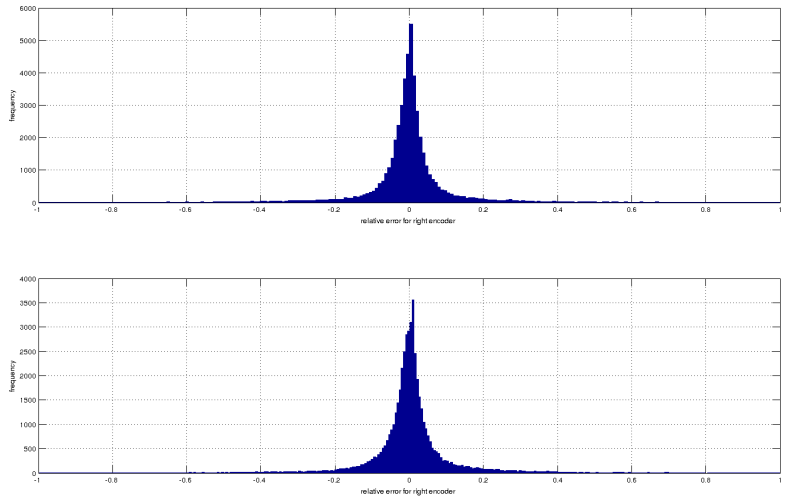


Figure 3.7: Distribution of relative error of the model trained via the HMC algorithm.

main memory. We used NetLab's MATLAB toolbox to set up the neural network's architecture and for the SCG and HMC procedure. The Bayesian optimization step was done using the "Spearmint" package from [58].

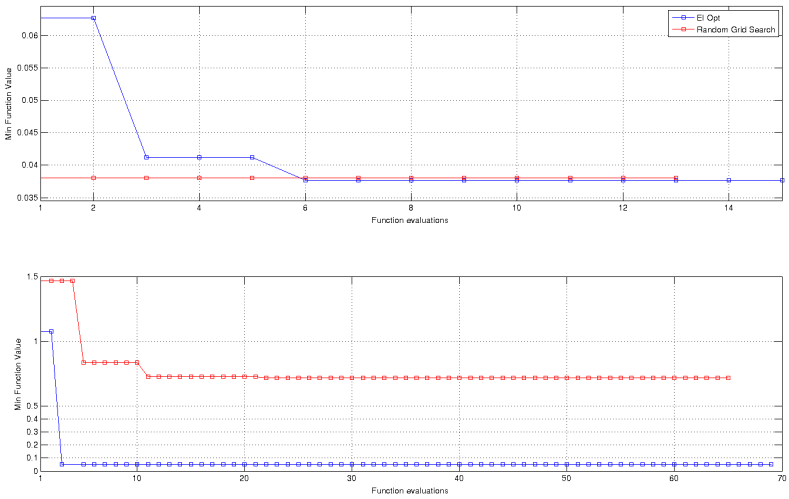


Figure 3.8: Comparison of expected improvement and random grid search as the acquisition function for optimizing the parameters of the deterministic model (top) and probabilistic model (bottom).

Chapter 4

Risk Assessment

Because the appropriate measure of and response to risk in a collaboratively controlled WC may be diagnosis and even user specific, our goal in this work is not to propose a particular approach to risk assessment, but rather to construct components from which a variety of assessments can be built. In this chapter we propose some ways in which such assessments can draw on the information in the model and map described earlier.

The fundamental tool for constructing a risk assessment in this framework is a simulation of the WC's future motion $x(\cdot)$ through the local map given a proposed future joystick signal $v(\cdot)$, where for convenience we will assume that both functions are discrete time with the same sampling period as that used to construct the model in chapter 3 (eg: 200 ms). The WC has nontrivial size, so at each sample of the simulation it will overlap with some subset of the map's grid cells, each of which has some likelihood of containing an obstacle. The design of a risk assessment therefore has a number of parameters to determine, including:

(i) simulation horizon,

(ii) number of times to evaluate risk over that horizon (could be as often as every 200 ms, although the WC will not typically move very much over such a short period),

(iii) how to account for the shape of the WC,

(iv) number of future joystick signals to consider, and

(v) what future joystick signals to consider.

Depending on available interventions and how the risk assessment will be used in these interventions, the form of the risk assessment might vary from a separate assessment for each possible input, a time dependent assessment, or perhaps just a single overall value. It is fairly clear that the reported risk should not decrease if the WC overlaps with a grid cell which is more likely to be occupied, if more of the WC overlaps with obstacles, if collision

is reported for more input signals and/or if a collision will occur sooner; however, these monotonicity constraints still leave considerable flexibility in choosing aggregation functions when more compact risk assessments are desired. Some examples of risk assessments which result in a single number:

- Greatest (log) likelihood of collision:

$$\text{risk} = \max_{v_{0:t_{\max}-1} \in V} \max_{0 \le t \le t_{\max}} \max_{\hat{x} \in \mathfrak{W}(x_t)} l_t(\hat{x}) \tag{4.1}$$

  where V is a set of input signals, $t_{\max}$ is the time horizon, $\mathfrak{W}(x(t))$ is the set of all points inside the WC when it is at point $x(t)$, and $l_t(\hat{x})$ is the log-odds of point $\hat{x}$ (or its nearest neighbour in the grid) being occupied by an obstacle according to the map at time t (as constructed in chapter 2). Typically only a subset of the time interval $[0, t_{\max}]$ would actually be tested, and $\mathfrak{W}(x(t))$ would be approximated by a sampling of the WC's shape.

- User weighted, time discounted likelihood of collision:

$$\text{risk} = \sum_{v(\cdot) \sim \mathcal{V}} \sum_{t=0}^{t_{\max}} \alpha^t \, p(v(\cdot)) \max_{\hat{x} \in \mathfrak{W}(x(t))} \exp(l_t(\hat{x}))$$

  where $\mathcal{V}$ is a discrete distribution of input signals approximating the user's future behaviour, $0 < \alpha < 1$ is a discount factor, and $p(v(\cdot))$ is the probability of input signal $v(\cdot)$.

- Time to collision:

$$\text{risk} = \operatorname*{argmin}_{t \in [0, t_{\max}]} \left[\int_y \psi(y, t, \mathfrak{W}(x(t))) \, l_t(y) \, dy > \hat{l} \right]$$

  where future state uncertainty is modelled by a spreading kernel function ψ rather than a sampling of input signals: only a single $v(\cdot)$ is used, but $\psi(y, t, W)$ is a time dependent kernel function over y (eg: $\int_y \psi(y, t, W) dy = 1$ for all t and any W) which starts at time $t = 0$ as an appropriately scaled indictator function for set W and gradually spreads out as t grows. In this case the risk is the first time at which the likelihood of collision (as measured by the integral) exceeds a threshold (controlled by the choice of $\hat{l}$). In practice, the integral would be approximated by quadrature or sampling.

In the next chapter we demonstrate our system using a version of the first risk assessment above, but there are many more possibilities. A Bayes' risk model [61], for example, have been used for robotic task execution under uncertainty. Domain-specific risk evaluation measures have also been applied to ensure task safety in human-robot dialog [62]. Althoff et al. [63] proposed a probabilistic model for safety assessment in dynamic environments under an imperfect sensing model.

Chapter 5

Experimental Results

We test the effectiveness of the described system in two scenarios. In the first, the system attempts to detect higher risk maneuvers when the user turns into a doorway from a corridor, a scenario meant to replicate a common source of collision observed in the trials described in [1] (as discussed in section 1). In the second, we demonstrate the effectiveness of the system as the user drives an extended trajectory through a cluttered environment while maneuvering the PWC in all directions. Note that the system is passively evaluating the risk metric in all of these experiments; the user (the author) is always in full control of the motion.

For the first scenario, the experimental setup uses a 914mm (36in) wide simulated doorway (a common width in North American public buildings). It should be noted that our PWC is 655mm wide and that health care facilities typically mandate wider doors—for example, 1168mm (46in) in the province of Ontario [64]—so our doorway poses a moderate degree of collision risk simply because of the relatively tight fit.

As shown in Figure 5.1, we perform four experiments:

(i) a safe slow turn,

(ii) an unsafe (early) slow turn,

(iii) a safe fast turn, and

(iv) an unsafe (early) fast turn.

In every experiment the PWC starts from rest approximately 2.3m down the corridor, and the user attempts to drive it parallel to the wall until the turn is initiated. For the "slow" trajectories, the speed of the PWC is capped at 50% of the maximum speed (∼0.4m/s) and the initial portion of the trajectory is ∼30cm from the corridor wall. For the "fast" trajectories the speed is 70–90% of maximum (∼0.7m/s) and the distance from the corridor wall is ∼64cm. A turning point was identified on the floor for each of the "unsafe" trajectories such that the PWC would impact about 7.5cm before the corner, although the PWC was stopped just before impact. For

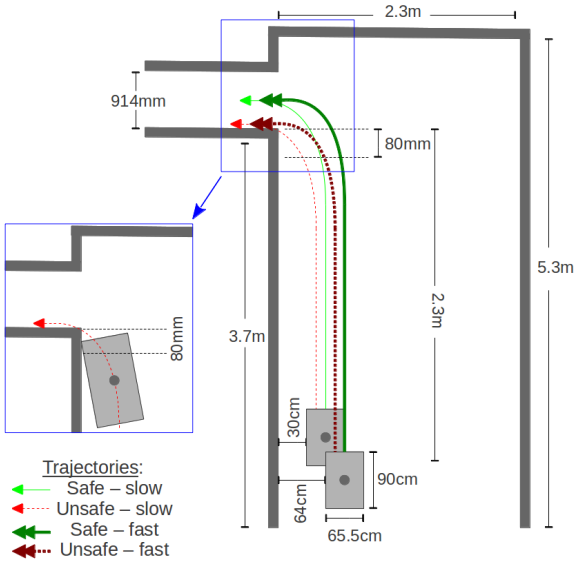


Figure 5.1: Experimental setup and the four tested trajectories.

the "safe" trajectories the turn was initiated approximately 15cm further along the corridor, so the PWC passed no closer than about 7.5cm from the corner, and was stopped after it had passed through the doorway. The right side of figure 2.4 shows a typical map status just before turning begins; the corner is not within the field of view of the camera and remains outside of it throughout the turn.

We report the greatest log-odds of collision risk metric (4.1). The horizon was chosen as $t_{\max} = 1.4$s. The set of future input signals V considered included only one: that the joystick was held constant at its current position. The PWC shape $\mathfrak{W}(x(t))$ is represented by 17 samples (5 along each of the long edges, 3 along the short edges and 1 in the center of a rectangular bounding box). The risk is recomputed every time a new joystick input arrives.

Each of the four experiments was repeated 10 times. Figures 5.2 and 5.3 show the time evolution of the risk metric (4.1) during each run of each experiment (time shifted so that the peak of every run for a single experiment is aligned). The scaling of the vertical axis is essentially arbitrary and the values of the metric have been capped to ± 10, so we are primarily interested in comparing the value of the metric between experiments rather

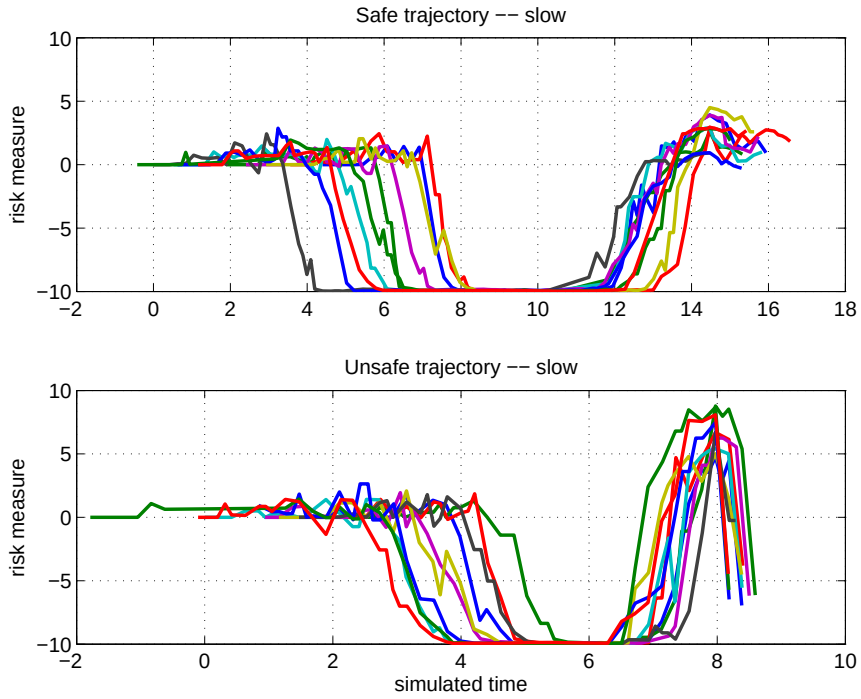


Figure 5.2: Safe (top) and unsafe (bottom) trajectories in slow-speed mode.

than drawing conclusions from its specific value.

All of the experimental runs follow the same basic pattern. When the system is first turned on at the start of each run, the risk metric hovers around zero until the sensors have time to determine that there is free space in front of the chair; we will consider this the "baseline" risk of driving through a region whose obstacles are unknown. At that point the metric drops significantly as the chair moves safely along the corridor parallel to the wall. When the chair begins its turn the metric rises quickly toward a peak, and then falls again when the chair is stopped at the end of the run.

The distinction between the four experiments lies in the height of the peak as the chair makes its turn. Consider first the "slow" experiments shown in figure 5.2. In the "slow unsafe" case on the bottom, all runs show the risk metric rising significantly above the baseline risk detected at the beginning of the run, indicating that a collision is a significant threat. In contrast, for the "slow safe" case on the top all runs show a distinctly lower peak. The fact that the chair is making a blind turn into a narrow doorway

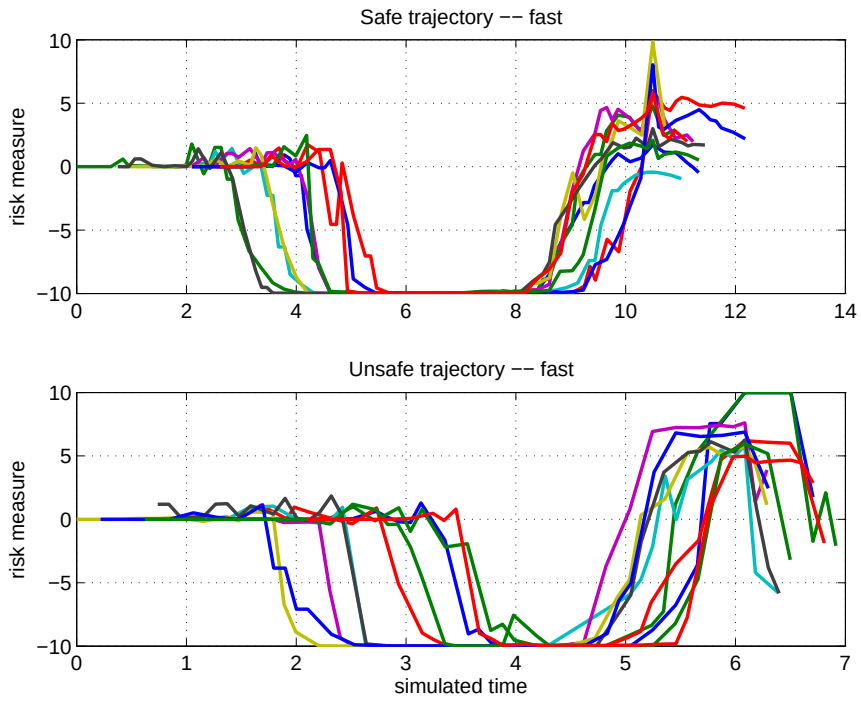


Figure 5.3: Safe (top) and unsafe (bottom) trajectories in fast-speed mode.

explains why these peaks are still at or slightly above the baseline risk at the beginning of the run: toward the end of the run, the chair is again predicting motion through unknown portions of the map. Now consider the "fast" experiments shown in figure 5.3. In the "fast unsafe" case, all runs show high peaks similar to those in the "slow unsafe" case, again indicating that the threat of collision is significant. In the "fast safe" case the range of peaks displayed by different runs is much wider—everything from baseline to the top of the scale—however, we are not terribly concerned that the risk may be overestimated in this case because most PWC users would consider a 90° turn at nearly full speed around a blind corner to be a somewhat dubious maneuver even if they were certain of avoiding the corner itself.

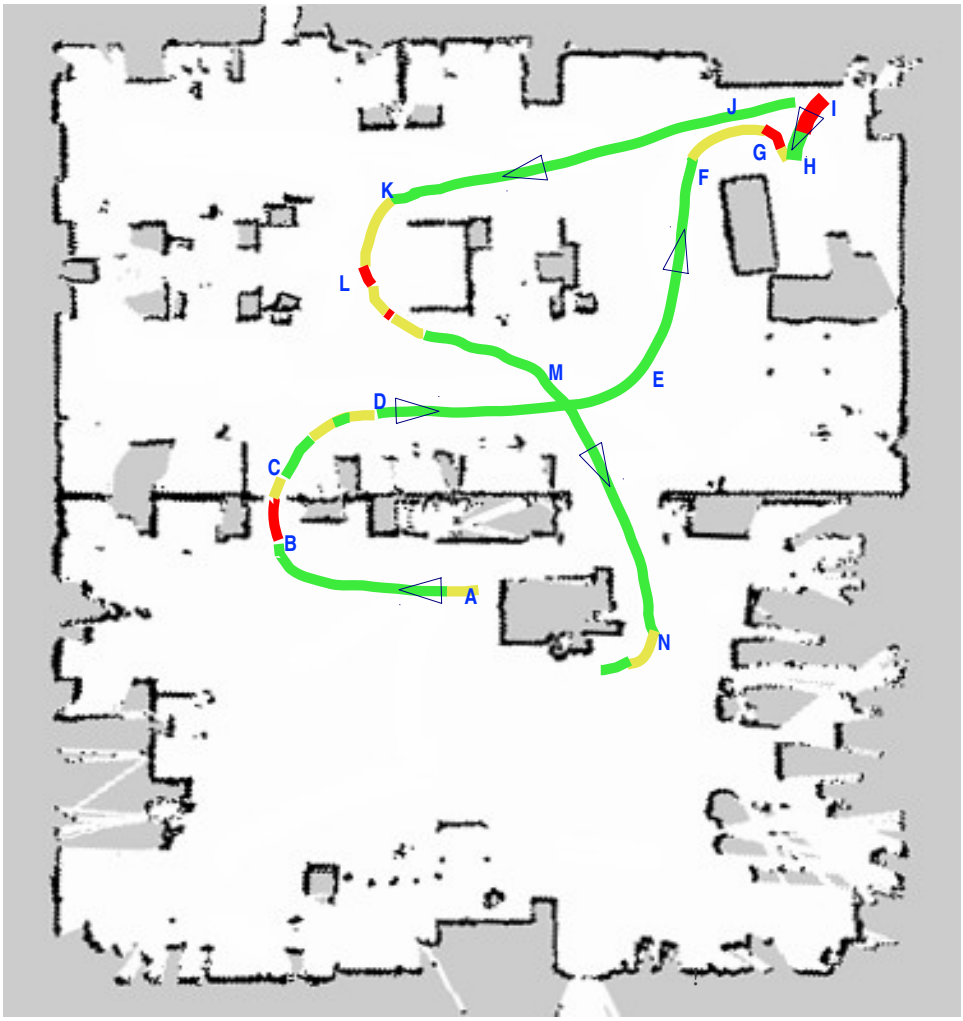


Figure 5.4: Occupancy-grid map of the environment navigated during the long-run trial.

In the second scenario the PWC is driven in a typical office environment, as depicted by the occupancy-grid map in Figure 5.4 (generated offline), along the path shown in colors and with arrows indicating wheelchair heading. The green lines indicate areas evaluated as not risky, yellow indicates areas of moderate risk, and red indicates areas of high risk along the path. The time-series plot of risk assessment along the path is shown in Figure 5.5.

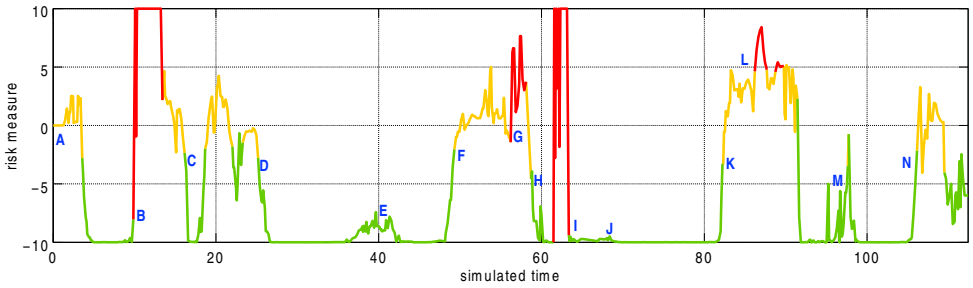


Figure 5.5: Time-series plot of risk assessment during the long-run trials shown in Figure 5.4.

We have added corresponding annotations (alphabetically ordered in time along the trajectory) to both figures to denote specific areas of interest. As can be seen, while travelling close to obstacles (such as at G and L) or through narrow doorways (such as between B and C), risk assessment is high. Of particular interest is the section of the map where the wheelchair performs a "backing up" maneuver between H and I with the wall directly behind it. This wall was detected by the camera and incorporated into the egocentric map while the chair was facing toward it between F and G; however, it is behind the chair and hence invisible to the camera while the chair is backing up between H and I. In spite of the obstacle being in the sensor's blindspot, the risk assessment goes from low to high as the chair approaches the wall while backing up after H, and then drops again when the chair stops backing at I in preparation for forward motion toward J. This reaction of the risk metric demonstrates the utility of our egocentric map in detecting potential collisions with previously observed obstacles now lying outside the camera's narrow field of view.

Chapter 6

Conclusions

In this thesis we discuss the construction of a dynamic egocentric occupancy map that is capable of maintaining information about local obstacles even when they are outside the field of view of the sensor. We construct a neural network model using the frequentist and Bayesian approach to map joystick inputs and wheelchair motion. We demonstrated that using the egocentric map and model infrastructure, we can evaluate a variety of risk assessment metrics for collaborative control of a smart wheelchair. One such metric is demonstrated on a wheelchair with a single RGB-D camera in a doorway traversal scenario where the near edge of the doorframe is no longer visible to the camera as the chair makes its turn.

6.1 Suggestions for Future Research

While there are approaches by which one can improve the mapping technique and modeling scheme, we dedicate this section to discussions of further improvement towards a smart PWC.

We demonstrated a naive risk assessment measure in which it only considered greatest log-odds of collision. Some future works can be dedicated to exploring (experimentally) the various risk assessment schemes similar to the schemes presented in chapter 4. While running experiments with (some) target population is a large undertaking, the process of designing the experiments is no less and requires great attention.

As explained in chapter 5, the only future input signal that was considered was based on the assumption that the joystick will be held constant at its current position. This is indeed a naive assumption. One could however "learn" future joystick movements from data. This can be a user specific model that is based on the driving habits of the user or based on data from population.

Since generally joystick commands reside in polar coordinates one could employ the properties of the egocentric polar map representation presented here to first solve for the configuration space in the obstacle map and then restrict joystick movements accordingly.

Bibliography

[1] P. Viswanathan, "Navigation and obstacle avoidance help (NOAH) for elderly wheelchair users with cognitive impairment in long-term care," Ph.D. dissertation, University of British Columbia, 2012.

[2] L. Fehr, W. E. Langbein, and S. B. Skaar, "Adequacy of power wheelchair control interfaces for persons with severe disabilities: A clinical survey," *Development*, vol. 37, no. 3, pp. 353–360, 2000.

[3] R. C. Simpson, E. F. LoPresti, and R. A. Cooper, "How many people would benefit from a smart wheelchair?" *Journal of Rehabilitation Research and Development*, vol. 45, no. 1, pp. 53–71, 2008.

[4] G. Pullin, *Design Meets Disability*. MIT Press, 2009.

[5] J. Miller Polgar, "The myth of neutral technology," in *Design and Use of Assistive Technology*, M. M. K. Oishi, I. M. Mitchell, and H. F. M. Van der Loos, Eds. Springer, 2010, pp. 17–23.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[7] P. Viswanathan, R. Wang, and A. Mihailidis, "Wizard-of-Oz and mixed-methods studies to inform intelligent wheelchair design for older adults with dementia," in *Association for the Advancement of Assistive Technology in Europe*, 2013.

[8] R. C. Simpson, "Smart wheelchairs: A literature review." *Journal of rehabilitation research and development*, vol. 42, no. 4, p. 423, 2005.

[9] B. M. Faria, L. P. Reis, N. Lau, J. C. Soares, and S. Vasconcelos, "Patient classification and automatic configuration of an intelligent wheelchair," in *Agents and Artificial Intelligence*. Springer, 2013, pp. 268–282.

[10] R. H. Wang, S. M. Gorski, P. J. Holliday, and G. R. Fernie, "Evaluation of a contact sensor skirt for an anti-collision power wheelchair for older adult nursing home residents with dementia: Safety and mobility," *Assistive Technology*, vol. 23, no. 3, pp. 117–134, 2011.

[11] A. C. Balcells and J. A. Gonz, "TetraNauta: A wheelchair controller for users with very severe mobility restrictions," in *Proc. 3rd TIDE Congress*, 1998, pp. 336–341.

[12] G. Bourhis, O. Horn, O. Habert, and A. Pruski, "An autonomous vehicle for people with motor disabilities," *Robotics & Automation Magazine, IEEE*, vol. 8, no. 1, pp. 20–28, 2001.

[13] E. Prassler, J. Scholz, and P. Fiorini, "A robotics wheelchair for crowded public environment," *Robotics & Automation Magazine, IEEE*, vol. 8, no. 1, pp. 38–45, 2001.

[14] E. B. Vander Poorten, E. Demeester, E. Reekmans, J. Philips, A. Huntemann, and J. De Schutter, "Powered wheelchair navigation assistance through kinematically correct environmental haptic feedback," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 3706–3712.

[15] Y. Wang and W. Chen, "Hybrid map-based navigation for intelligent wheelchair," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 637–642.

[16] G. Peinado, C. Urdiales, J. Peula, M. Fdez-Carmona, R. Annicchiarico, F. Sandoval, and C. Caltagirone, "Navigation skills based profiling for collaborative wheelchair control," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 2229–2234.

[17] T. Carlson and Y. Demiris, "Increasing robotic wheelchair safety with collaborative control: Evidence from secondary task experiments," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 5582–5587.

[18] Y. Satoh and K. Sakaue, "An omnidirectional stereo vision-based smart wheelchair," *EURASIP Journal on Image and Video Processing*, vol. 2007, no. 1, p. 087646, 2007.

[19] R. Simpson, E. LoPresti, S. Hayashi, I. Nourbakhsh, D. Miller *et al.*, "The smart wheelchair component system," *Journal of Rehabilitation Research and Development*, vol. 41, no. 3B, pp. 429–442, 2004.

[20] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[21] J. Borenstein and Y. Koren, "Obstacle avoidance with ultrasonic sensors," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 213–218, 1988.

[22] ——, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[23] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. 3, no. 3, pp. 249–265, 1987.

[24] D. Kortenkamp, R. P. Bonasso, and R. Murphy, Eds., *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, 1st ed. AAAI Press, March 1998.

[25] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[26] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun, "Sonar-based mapping of large-scale mobile robot environments using EM," in *Proc. of the 16th International Conference on Machine Learning*, 1999.

[27] S. Thrun, W. Burgard, and D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Autonomous Robots*, vol. 5, no. 3-4, pp. 253–271, 1998.

[28] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, 1997, pp. 146–151.

[29] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2008.

[30] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[31] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[32] S. Thrun, "Learning occupancy grids with forward sensor models," *Autonomous Robots*, vol. 15, pp. 111–127, 2002.

[33] J.-H. Park, Y.-D. Shin, J.-H. Bae, and M.-H. Baeg, "Spatial uncertainty model for visual features using a Kinect sensor," *Sensors*, vol. 12, no. 7, pp. 8640–8662, 2012.

[34] T. Petrinic, E. Ivanjko, and I. Petrovic, "AMORsim - a mobile robot simulator for matlab," in *Proceedings of 15th International Workshop on Robotics in Alpe-Adria-Danube Region, Balatonfüred, Hungary*, 2006.

[35] S. Suddarth, S. Sutton, and A. Holden, "A symbolic-neural method for solving control problems," in *IEEE International Conference on Neural Networks*, 1988, pp. 516–523 vol.1.

[36] L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1990, pp. 396–404.

[37] D. J. MacKay, "Bayesian non-linear modeling for the prediction competition," in *Maximum Entropy and Bayesian Methods*. Springer, 1996, pp. 221–234.

[38] S. C. Suddarth and A. D. Holden, "Symbolic-neural systems and the use of hints for developing complex systems," *International Journal of Man-Machine Studies*, vol. 35, no. 3, pp. 291 – 311, 1991.

[39] M. Nagrath and I. Nagrath, *Robotics and Control*. Tata McGraw-Hill, 2003.

[40] C. Andrieu, N. D. Freitas, and A. Doucet, "Robust full Bayesian learning for neural networks," 1999.

[41] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 5, no. 4, pp. 455–455, 1992.

[42] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481–1497, 1990.

[43] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[44] R. M. Neal, "Bayesian learning for neural networks," Ph.D. dissertation, University of Toronto, 1995.

[45] J. A. Anderson and E. Rosenfeld, Eds., *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988.

[46] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993.

[47] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 111–147, 1974.

[48] Y. Zweiri, J. Whidborne, and L. Seneviratne, "A three-term backpropagation algorithm," *Neurocomputing*, vol. 50, no. 0, pp. 305 – 318, 2003.

[49] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," pp. 409–436, 1952.

[50] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Carnegie Mellon University, Tech. Rep., 1994.

[51] I. Nabney, *NETLAB: Algorithms for Pattern Recognition*, ser. Advances in Computer Vision and Pattern Recognition. Springer, 2002.

[52] R. Neal, "Bayesian training of backpropagation networks by the Hybrid Monte Carlo method," University of Toronto, Tech. Rep., 1993.

[53] D. J. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural computation*, vol. 4, no. 3, pp. 448–472, 1992.

[54] W. L. Buntine and A. S. Weigendt, "Bayesian back-propagation," *Complex Systems*, vol. 5, pp. 603–643, 1991.

[55] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[56] S. Duane, A. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Physics Letters B*, vol. 195, no. 2, pp. 216 – 222, 1987.

[57] S. M. Ziyu Wang and N. de Freitas, "Adaptive Hamiltonian and Riemann Manifold Monte Carlo samplers," Technical Report, Tech. Rep.

[58] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012.

[59] B. Leimkuhler and S. Reich, *Simulating Hamiltonian Dynamics*. Cambridge University Press, 2005.

[60] C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan, "An introduction to MCMC for machine learning," *Machine Learning*, vol. 50, no. 1-2, pp. 5–43, 2003.

[61] F. Doshi, J. Pineau, and N. Roy, "Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs," in *Proceedings of the 25th International Conference on Machine Learning*. ACM New York, NY, USA, 2008, pp. 256–263.

[62] J. Sattar and G. Dudek, "Towards quantitative modeling of task confirmations in human–robot dialog," in *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*, Shanghai, China, May 2011, pp. 1957–1963.

[63] D. Althoff, J. Kuffner, D. Wollherr, and M. Buss, "Safety assessment of robot trajectories for navigation in uncertain and dynamic environments," *Autonomous Robots*, vol. 32, no. 3, pp. 285–302, 2012.

[64] Ontario Ministry of Health and Long-Term Care, "Long-term care facility design manual," 1999.