Active Robot Perception using Programmable Light Curtains

Siddharth Ancha

August 2022
CMU-ML-22-104

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis committee

David Held	CMU *(Co-chair)*
Srinivasa Narasimhan	CMU *(Co-chair)*
Katerina Fragkiadaki	CMU
Wolfram Burgard	TU Nuremberg

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

This work was supported by the National Science Foundation under grants IIS-1849154, IIS-1900821, CPS-2038612 and by the United States Air Force and DARPA under contract FA8750-18-C-0092.

Keywords: Active Perception, Robotics, Light Curtains, Machine Learning, Computer Vision

To Mandy and my family

Abstract

Most 3D sensors used in robot perception, such as LiDARs, *passively* scan the entire environment while being decoupled from the perception system that processes the sensor data. In contrast, *active perception* is an alternative paradigm for robotics where a *controllable* sensor adaptively focuses its sensing capacity only on the most useful regions of the environment. Programmable light curtains are a recently-invented, resource-efficient, active sensor that measure the depth of any user-specified surface ("curtain") at a significantly higher resolution than LiDAR. The main research challenge is to design perception algorithms that decide where to place the light curtain at each timestep, tightly coupling sensing and control in a closed loop.

This thesis lays the algorithmic foundations for active robot perception using programmable light curtains. We investigate the use of light curtains for various perception tasks such as 3D object detection, depth estimation, obstacle detection and avoidance, and velocity estimation. First, we incorporate the velocity and acceleration constraints of the light curtain into a *constraint graph*; this allows us to compute feasible light curtains which optimize any task-specific objective. Then, we develop a suite of algorithms using a variety of tools such as Bayesian inference, deep learning, information gain and dynamic programming to intelligently place light curtains in the scene.

Finally, we combine multiple intelligent placement strategies in an online learning framework. First, we are able to explicitly estimate velocities and positions of scene points using a Bayes filtering technique based on particle filters and occupancy grids. Then, we propose a novel self-supervised reward function that evaluates the accuracy of current velocity estimates using future light curtain placements. This insight enables an online multi-armed bandit framework to intelligently switch between multiple placement policies in real time, outperforming individual policies. These algorithms pave the way for controllable light curtains to accurately, efficiently and purposefully perceive complex and dynamic environments.

Acknowledgements

I would first like to thank my advisors, David Held and Srinivasa Narasimhan without whom this thesis would not have been possible. Their guidance has immensely influenced my research and my thinking. I am greatful for their advice, support and mentorship.

I started working with Dave in the fall of 2017, and the journey has been incredible! From Dave I have learned the arts of critical thinking, experiment design and analysis, mantaining a thorough research journal and technical writing, among many other valuable skills. Dave has been very supportive of my research ideas and inclinations, including a major shift in my research direction after my second year. He has been exceptionally helpful and proactive in my professional development and providing career advice. It has been an absolute pleasure working with Dave!

I started working with Srinivas in the fall of 2019. His invention of programmable light curtains provided me a unique opportunity to work on the exciting problem of active perception at the intersection of computational imaging, computer vision, machine learning, planning and control that constitutes this thesis. From Srinivas I have learned how to think about high-level research problems and directions, and the arts of visualization and effective presentation. It has been a lot of joy and fun to work with Srinivas!

I would like to thank Katerina Fragkiadaki and Wolfram Burgard for taking the time out of their busy schedules to serve on my Ph.D. thesis committee. I am grateful for their valuable feedback, suggestions and advice on my research that helped improve my thesis.

I'm grateful for having the opportunity to collaborate with Ji Zhang in the final stages of my Ph.D. I have enjoyed and learned a lot from using his autonomous exploration development environment and his wheelchair robot platform.

I want to especially thank my collaborators Gaurav Pathak, Yaadhav Raaj, Peiyun Hu, Joe Bartels, Jianren Wang, Junyu (Jenny) Nan, Jianing (Aurora) Qian, Robert Tamburo, Haiming Gang, Yi-Ting Chen, Xiaochen Han and Tanay Sharma. Working with them has been a remarkable experience and a highlight of my Ph.D.

I would like to thank my colleagues in Dave's Robots Perceiving and Doing (R-PAD) lab: Ben Eisner, Xingyu Lin, Brian Okorn, Chuer Pan, Daniel Seita, Harshit Sikchi, Yufei Wang, Thomas Weng and Wenxuan Zhou. Although I did not get the opportunity to collaborate, I learned a lot from them and I cherish our discussions and time spent together.

I would like to thank my colleagues at Srinivas' Illumination and Imaging (ILIM) lab: Bowei Chen, Adithya Pediredla, Dinesh Reddy, Mark Sheinin, Shumian Xin and Tianyuan Zhang. Even though we worked on very different projects, I had fun sharing lab space, having research discussions, and streaming the NBA finals on the large screen.

I would like to thank members of the biweekly light curtain meetings: Henny Admoni, Matt O'Toole, Abhijat Biswas and Gustavo Silvera. I learned a lot discussing their very interesting projects, and I thank them for providing feedback on my research and practice talks.

I am grateful for the research environment at CMU and all my amazing friends and colleagues including Hitesh Arora, Shaojie Bai, Sebastián Caldas, Devendra Chaplot, Ankush Das, Arjav Desai, Ben Eysenbach, Anjalie Field, Dhiraj Gandhi, Kartik Gupta, Mansi Gupta, William Guss, Kin Gutiérrez, Brandon Houghton, Leonid Keselman, Lisa Lee, Jin Li, Kevin Lin, Leqi Liu, Tanya Marwah, Stephanie Milani, Shohin Mukherjee, Adithya Murali, Lerrel Pinto, Gregory Plumb, Biswajit Paria, Shrimai Prabhumoye, Danish Pruthi, Aaron Rumack, Kanthashree Mysore Sathyendra, Pratyusha Sharma, Jagjeet Singh, Nicholay Topin, Stefania La Vatiatta, Yufei Ye, Chih-Kuan (Jason) Yeh and Han Zhao. The spring break in the Amazon rainforest with my MLD friends was one of the most memorable trips of my life. I would like to thank Diane Stidle, my Ph.D. program administrator, for her immense help, support and patience throughout the last five years.

I would like to give special thanks to my partner, Amanda Coston. I met Mandy during my Ph.D., and I consider this my greatest achievement at CMU. She has been a constant source of love, joy and support during my Ph.D. Her company ironically made the COVID-19 pandemic a cherishable experience for me, and it is impossible to imagine enduring the pandemic without her. Navigating our Ph.D.s together made mine significantly more smooth and fun. I have learned a lot from her, and I cannot overstate how grateful I am for having Mandy by my side during this journey.

Another special thanks goes to our cute and fluffy dog, Tiggy. Tiggy has been the center of our lives, and has been my steadfast office-mate when working from home. After a hard day of work, nothing can be more comforting than being greeted by Tiggy with a toy in his mouth and his wagging tail.

I want to thank my family: my mother, father, and my brother Samarth, for their constant love, support and encouragement throughout my graduate studies despite being halfway across the globe. I would not have done enrolled in a Ph.D. program without their support, and I owe all my accomplishments to them.

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

3D sensors, such as LiDAR, have become ubiquitous for perception in autonomous systems operating in the real world, such as self-driving vehicles and field robots. Combined with recent advances in deep-learning based visual recognition systems, they have lead to significant breakthroughs in perception for navigation and autonomous driving, enabling the recent surge of commercial interest in self-driving technology.

However, most 3D sensors in use today perform *passive perception*, meaning that they continuously sense the entire environment while being completely decoupled from the recognition system that will eventually process the sensor data. In such a case, sensing the entire scene can be potentially inefficient. For example, consider an object detector running on a self-driving car that is trying to recognize objects in its environment. Suppose that it is confident that a tree-like structure on the side of the street is not a vehicle, but it is unsure whether an object turning around the curb is a vehicle or a pedestrian. In such a scenario, it might be beneficial if the 3D sensor focuses on collecting more data from the latter object, rather than distributing its sensing capacity uniformly throughout the scene.

Given a LiDAR sensor, the locations of obstacles can be computed from the captured point cloud; however, LiDARs are typically expensive, low-resolution and slow. Cameras are cheaper and high-resolution and 2D depth maps of the environment can be predicted from the images. However, depth estimation from camera images is prone to errors. They cannot guarantee safety due to issues like oversaturation, feature correspondence errors and scale ambiguity.

An alternative approach is to use *active perception* [Baj88, BAT18], where only the important and required parts of the scene are accurately sensed. In this work, we propose using a sensor that can perform active perception, i.e. a sensor that can be purposefully controlled to sense specific regions in the environment in an intelligent manner. Programmable light curtains [WBW$^+$18, BWWN19] were recently proposed as *controllable*, light-weight, and resource efficient sensors that measure the presence of objects intersecting any vertical 2D ruled surface (or a ‘curtain‘) whose shape can be specified by the user (see Fig. 3.1). There are three main advantages of using programmable light curtains over LiDARs. First, they can

Visit the CMU light curtains landing page for webpages on various components of this thesis: https://www.cs.cmu.edu/~ILIM/light_curtains. We have also released open source code repositories related to various components of this thesis at the following Github organization: https://github.com/CMU-Light-Curtains.

be cheaply constructed, since light curtains use ordinary CMOS sensors (a current lab-built prototype costs \$1000, and the price is expected to go down significantly in production). In contrast, a 128-beam Velodyne LiDAR that is commonly used in 3D self-driving datasets like KITTI [GLSU13] costs upwards of \$100,000. Second, light curtains generate data with much higher resolution in regions where they actively focus [BWWN19] while LiDARs sense the entire environment and have low spatial and angular resolution. Finally, light curtains are significantly faster than LiDARs. They can operate at 60 Hz whereas LiDARs typically operate between 5-20 Hz.

Because they use an ordinary rolling shutter camera, light curtains combine the best of both worlds of passive cameras (high spatial-temporal resolution and lower cost) and LiDARs (accurate detection along the 2D curtain and robustness to scattered media like smoke/fog).

One weakness of light curtains is that they are able to sense only a subset of the environment – a vertical ruled surface (see Fig. 5.1(c,d), Fig 3.1). In contrast, a LiDAR senses the entire scene. To mitigate this weakness, we can take advantage of the fact that the light curtain is a *controllable* sensor – we can choose where to place the light curtains. Thus, we must intelligently place light curtains in the appropriate locations, so that they sense the most important parts of the scene.

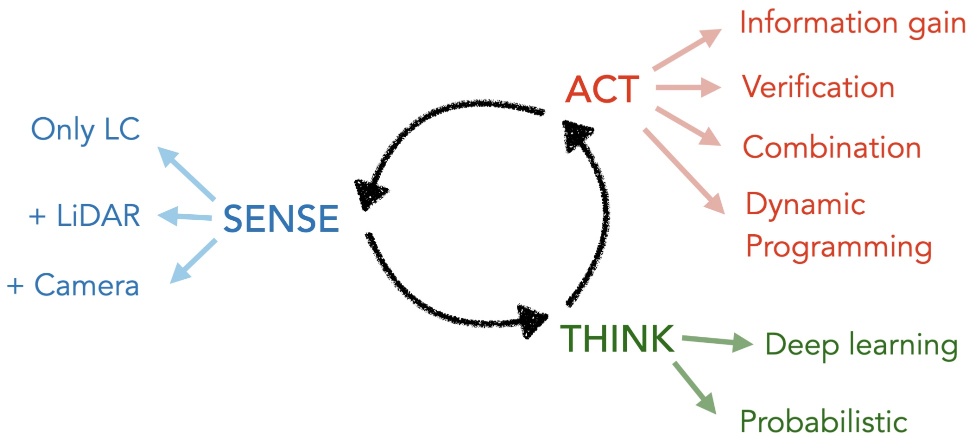


Figure 1.1: **Active perception pipeline and methods proposed in this thesis.** In this thesis, we develop active perception systems that have three components. (1) SENSE: sensing using light curtains in isolation or in conjuction with additional sensors such as LiDARs or RGB camera. (2) THINK: involves inferring and predicting the state of the world from sensor observations. We propose deep learning methods as well as probabilistic Bayesian methods for inference. Finally, (3) ACT: planning and decision making to decide where to place the next light curtain. We place light curtains either at the regions of highest uncertainty to maximize information gain, place curtains to verify our predicted object locations, or combine multiple different placement strategies in intelligent ways. We also dynamic programming to ensure that the computed curtains satisfy physical constraints of the device. The three processes are inter-dependent and inter-connected in a closed loop.

In this thesis, we develop the algorithmic foundations of active perception systems using programmable light curtains. The perception systems we develop all have three main components as shown in Fig. 1.1. The three components are highly inter-dependent and inter-connected in a closed loop:

1. **SENSE**: Sensing using light curtains. Light curtains can be used alone, or in conjuction with other conventional sensors such as LiDARs or cameras.

2. **THINK**: This involves inferring and predicting the state of the world from sensor observations. We propose both deep learning based methods as well as probabilistic Bayesian inference methods for prediction and state estimation.

3. **ACT**: This involves planning and decision making to decide where to place the next light curtain, based on the inference/predictions from the THINK component. We place light curtains either at the regions of highest uncertainty to maximize information gain, to verify predicted object locations by placing them at those locations, or intelligently combining multiple curtain placement stratgies. We also develop dynamic programming based planning methods to ensure that computed curtains are feasible and satisfy the velocity and acceleration constraints of the light curtain device. The computed curtains are sent to the SENSE component for imaging.

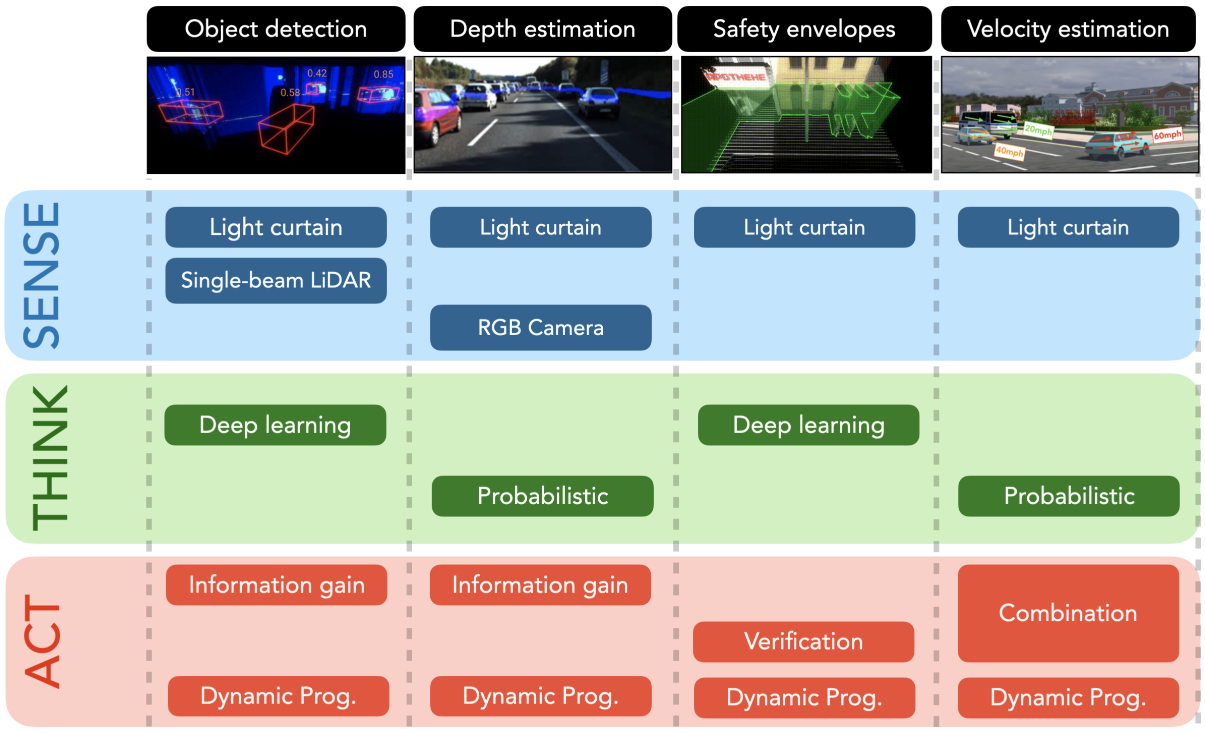


Figure 1.2: *Columns:* the four active perception tasks investigated in this thesis. *Rows:* the methods we propose for each of the SENSE, THINK and ACT components of the active perception pipline in order to solve each task. This thesis explores a diverse set of tasks and a diverse set of approaches for sensing, inference and decision-making using programmable light curtains.

We apply the above methods to various active perception tasks, as outlined in Fig. 1.2:

- **Active object detection** [Chapter 5]: We propose using light curtain measurements in conjuction with a single-beam LiDAR (a sparse but accurate sensor) to detect 3D bounding boxes around objects in a scene. We train a deep-learning based 3D object detector to detect objects of the required semantic classes and output bounding boxes. We place curtains at the locations of highest detector uncertainty to maximize information gain and sense more 3D points. This incrementally and consistently improves detection performance.

- **Active depth estimation** [Chapter 6]: We perform depth estimation of a scene by combining measurements from a light curtain and a monocular RGB camera (a dense but inaccurate sensor) using a Bayesian inference framework. A monocular depth estimator is used to input an RGB image and output a depth distribution. We treat this distribution as a prior and combine it with likelihoods from light curtain measurements to refine the monocular depth estimates.
- **Active safety envelopes** [Chapter 7]: A "safety envelope" is a hypothetical surface that separates a robot from all obstacles in its environment while being as close to the object surfaces as possible. The task is to estimate the safety envelope of the scene at each timetep. We train a deep neural network to predict the location of the safety envelope in the next timestep from previous light curtain measurements alone. Then, we place "verification" curtains at the predicted object locations to sense visible surfaces and estimate and track the safety envelope across time.
- **Active velocity estimation** [Chapter 8]: We develop a method to explicitly predict the velocity of each point in the scene using light curtains. We use a probabilistic Bayes filtering approach using dynamic occupancy grids that combine conventional occupancy grids with particle filters. After estimating the velocity and occupancy of each location in the scene, we combine the information-gain based and verification-based placement strategies. We use a multi-armed bandit framework to intelligently switch between multiple strategies. This is enabled by a novel self-supervised reward function computed at test time using light curtain placements alone, improving the accuracy of velocity and occupancy estimation.

Constraint graph and dynamic programming [Chapter 4]: We develop a method to incorporate the velocity and acceleration constraints of the light curtain device into a "constraint graph". Then we develop a planning algorithm using dynamic programming that efficiently computes the feasible curtain that maximizes any task-specific objective while satisfying light curtian constraints. Furthermore, we show how "random curtains" can be sampled from the constraint graph to discover obstacles with very high probability. We develop another dynamic programming based algorithm that analytically computes the probability of random curtains discovering obstacles, providing theoretical safety guarantees for obstacle detection and avoidance. The constraint graph and dynamic programming are used by all active perception tasks based on light curtains (included in the ACT component).

Structured vs unstructured environments: The focus of this thesis is on developing active perception methods for *unstructured* environements. An example of an unstructured environment is a robot navigating an indoor building with an unknown floor plan. The building may be crowded with human obstacles who walk in arbitrary and haphazard trajectories. The techniques and algorithms we develop do not assume any prior knowledge about the environment – the types of objects that may be present, the layout of the environment or patterns of object motion. However, in highly structured settings like urban driving, it may be beneficial to exploit the structure present in the environment. For example, light curtains may be placed along lane markings to detect collisions from vehicles in adjacent lanes, or on crosswalks to detect pedestrians. We do not focus on structured environments for two reasons. First, structure is very environment-specific and may vary considerably between environements. For example, the structure in urban driving is very different from fulfillment robots operating in a warehouse. Second, most real environments are never *perfectly* structured. A self-driving vehicle could encounter unexpected events, such as pedestrians running onto the street or vehicles breaking traffic laws. In order for robots to navigate safely, they must be prepared for structure to break down anytime. Therefore, the techniques developed in this thesis are applicable, and essential, for active light curtain perception in both structured and unstructured environments.

Websites and code release: We have created websites and released code for various components of this thesis. All websites can be found on the CMU light curtains landing page. All codebases are hosted on a Github organization.

- CMU light curtains landing page: [Landing webpage[1]]
- Github organization for code: [Github organization[2]]

The Github organization and landing page contain material related to each of the following components:

- **Object detection** [Website[3]] [Code[4]]: Active object detection using light curtains.
- **Depth estimation** [Website[5]] [Code[6]]: Active depth estimation using light curtains.
- **Safety envelopes** [Website[7]] [Code[8]]: Active obstacle detection and avoidance using light curtains.
- **Contraint graph** [Code[9]]: Library containing the implementation of optimization using the light curtain constraint graph.
- **Light Curtain Simulator** [Code[10]]: Code for simulating programmable light curtains.

[1] https://www.cs.cmu.edu/~ILIM/light_curtains
[2] https://github.com/CMU-Light-Curtains
[3] https://siddancha.github.io/projects/active-perception-light-curtains/
[4] https://github.com/CMU-Light-Curtains/ObjectDetection
[5] https://soulslicer.github.io/rgb-lc-fusion/
[6] https://github.com/CMU-Light-Curtains/DepthEstimation
[7] https://siddancha.github.io/projects/active-safety-envelopes-with-guarantees/
[8] https://github.com/CMU-Light-Curtains/SafetyEnvelopes
[9] https://github.com/CMU-Light-Curtains/ConstraintGraph
[10] https://github.com/CMU-Light-Curtains/Simulator

CHAPTER 2

Related Work

2.1 Active perception

Active Perception encompasses a variety of problems and techniques that involve actively controlling the sensor for improved perception [Baj88, BAT18, Wil94]. Examples include actively modifying camera parameters [Baj88], moving a camera to look around occluding objects [CAF18], and obtaining the next-best-view [Con85]. Prior works have used active perception for static scenes [MHG+14, BMK14] via a series of controllable partial glimpses. Our paper differs from past work because we use a controllable depth sensor (light curtains) and combine it with deep learning uncertainty estimates in a novel active perception algorithm.

2.2 Programmable light curtains

Programmable light curtains were introduced in prior work [WBW+18, BWW+19] as an adaptive depth sensor. Prior work has also explored the use of light curtains. Ancha et. al. [ARH+20] introduced the light curtain constraint graph to compute feasible light curtains. Bartels et. al. [BWW+19] were the first to empirically use random curtains to quickly discover objects in a scene. However, there are several key differences from our work. First, we solve a very different problem: while Ancha et. al. [ARH+20] use light curtains to perform active bounding-box object detection in static scenes, whereas we track the safety envelope of scenes with dynamic objects. Although we build upon their constraint graph framework, we make several significant and novel contributions. Our main contribution is the safety analysis of random light curtains, which uses dynamic programming (DP) to produce theoretical guarantees on the probability of discovering objects. Providing theoretical guarantees is essential to guarantee safety, and is typically a hard task for perception systems. These works [ARH+20, BWW+19] do not provide any such guarantees. Additionally, we extend its constraint graph (that previously encoded only velocity constraints) to also incorporate acceleration constraints. Finally, we combine the discovery of safety envelopes using random curtains, with an ML approach that efficiently forecasts and tracks the envelope; this combination is novel, and we show that our method outperforms other approaches on this task.

2.3 Object detection from point clouds

There have been many recent advances in deep learning for 3D object detection. Approaches include representing LiDAR data as range images in LaserNet[MLK+19], using raw point clouds [SWL19], and using point clouds in the bird's eye view such as AVOD [KML+18], HDNet [YLU18] and Complex-YOLO [SMAG18]. Most state-of-the-art approaches use voxelized point clouds, such as VoxelNet [ZT18], PointPillars [LVC+19], SECOND [YML18], and CBGS [ZJZ+19]. These methods process an input point cloud by dividing the space into 3D regions (voxels or pillars) and extracting features from each of region using a PointNet [QSMG17] based architecture. Then, the volumetric feature map is converted to 2D features via convolutions, followed by a detection head that produces bounding boxes. We demonstrate that we can use such detectors, along with our novel light curtain placement algorithm, to process data from a single beam LiDAR combined with light curtains.

2.4 Next-best view planning

Next-best view (NBV) planning refers to a broad set of problems in which the objective is to select the next best sensing action in order to solve a specific task. Typical problems include object instance classification [WSK+15, DKMK16, DB02, SRR03] and 3D reconstruction [ISDS16, KRBS15, VGSMCLD14, DC17, HH12]. Many works on next-best view formulate the objective as maximizing information gain (also known as mutual information) [WSK+15, DB02, ISDS16, KRBS15, VGSMCLD14, DC17], using models such as probabilistic occupancy grids for beliefs over states [WSK+15, ISDS16, KRBS15, VGSMCLD14, DC17]. Our method is similar in spirit to next-best view. One could consider each light curtain placement as obtaining a new view of the environment; we try to find the next best light curtain that aids object detection. In Sec. 5.2.3 and Appendix 5.4, we derive an information-gain based objective to find the next best light curtain placement.

2.5 Depth estimation from RGB images

Depth from monocular and multi-Camera RGB has been extensively studied. We focus on a class of probabilistic depth estimation approaches that have reformulated the problem as a prediction of per-pixel depth distribution [LGK+19] [YHR19] [CC18] [YLL+18] [ICG+18] [XSC19]. Some of this work has passively exploited and refined [LGK+19] [XSC19] the uncertainty in the depth values via moving cameras and multi-view camera constraints, but have not used the capabilities of adaptive densors available.

There is a large body of prior work on depth estimation across multiple frames [LGK+19, ZSL+19, CKBP18, ZGW+18, MSK, WPF19, PVGDVG20]. Liu et. al. [LGK+19] aggregate per-frame depth estimates across frames using Bayesian filtering. Matthies et. al. [MSK] use a similar Bayesian approach, but their method is only applied to controlled scenes and restricted camera motion. Other works [ZSL+19, CKBP18, WPF19, PVGDVG20] use RNNs for predicting depth maps at each frame. All of aforementioned works try to predict the full 2D depth map of the environment from monocular images.

2.6 Depth from 3D sensors

Fixed-scan 3D sensors use a fixed scan light source / receiver to perceive depth. Long range outdoor depth from these such as commercially available time-of-flight (TOF) cameras or LiDARs provide dense metric depth with confidence values with wide usage in research [GLSU13, CBL$^+$20, CLS$^+$19]. However, beside low resolution, these sensors can be prohibitively expensive.

Adaptive sensors use a dynamically controllable light source / receiver instead. These have been making headway in long-range outdoor applications. Adaptive sensing via focal length/baseline variation through the use of servos and motors [MCCY18, GFMP08, NMH$^+$05, SST18], directionally controlled time-of-flight ranging using a MEMS mirror / laser [TWXS18, TJFK20, YKI$^+$18], gated depth imaging [WGRD20, GJAB$^+$19, GS15] and sampling specific depth profiles using programmable light curtains [BWW$^+$19, WBW$^+$18, ARH$^+$20] are just some examples. However, these methods do not seem to exploit or fuse data from RGB modalities yet. Various work by Bergman, Nishimura et. al. [BLW20, NLMW20] and Pittaluga et. al. [PTF$^+$20] present sensors and algorithms for adaptive sensing via 2D angular sampling, providing precise depth at limited number of pixels. However, our light curtain approach performs depth sampling via adaptive depth gating, giving useful information at every pixel at a higher resolution. Nishimura's sensor uses a SPAD where light is spread out over the entire field of view, limiting its range and operation outdoors in ambient light. Light curtains however, maximize the light energy on the region of interest via triangulation.

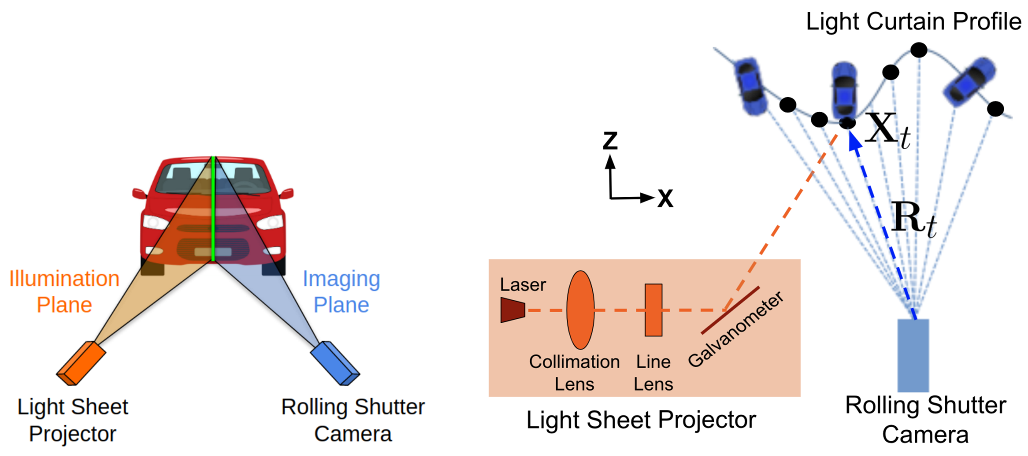
2.7 Safe navigation

Many approaches for safety guaranteed navigation use 3D sensors like LiDARs [RVBR18, RWR14, SAMR18] and/or cameras [RR17, BBB$^+$19]. The sensor data is converted to occupancy grids/maps [RVBR18, RWR14, BBB$^+$19]; safety and collision avoidance guarantees are provided for planning under these representations. Other works use machine learning models to recognize unsafe, out-of-distribution inputs [RR17] or learning to predict collision probabilites [RVBR18, RWR14]. Our work of estimating the safety envelope using a light curtain is orthogonal to these works and can leverage those methods for path planning and obstacle avoidance.

CHAPTER 3

Background on Light Curtains

Programmable *light curtains* [WBW+18, BWWN19] are a sensor for adaptive depth sensing. "Light curtains" can be thought of as virtual surfaces placed in the environment. They can detect points on objects that intersect this surface. The working principle is illustrated in Fig. 3.1(a, b).



(a) Working principle (b) Optical schematic (top view)

Figure 3.1: Illustration of programmable light curtains adapted from [BWWN19, WBW+18]. (a) An illumination plane (from the projector) and an imaging plane (of the camera) intersect to produce a light curtain. (b) A controllable galvanometer mirror rotates synchronously with a rolling shutter camera and images the points of intersection, $\mathbf{X}_t$ (called control points).

3.1 Coordinate system

Before explaining how the curtain is created, we briefly describe our coordinate system and the basics of a rolling shutter camera. Throughout the paper, we will use the standard camera coordinate system centered at the sensor. We assume that the z axis corresponds to depth from the sensor pointing forward, and that the y vector points vertically downwards. Hence the xz-plane is parallel to the ground and corresponds to a top-down view, also referred to as the bird's eye view.

3.2 Rolling shutter camera

A rolling shutter camera contains pixels arranged in T number of vertical columns. Each pixel column corresponds to a vertical imaging plane. Readings from only those visible 3D points that lie on the imaging plane get recorded onto its pixel column. We will denote the xz-projection of the imaging plane corresponding to the t-th pixel column by ray R_t, shown in the top-down view in Fig. 3.1(b). We will refer to these as "camera rays". The camera has a rolling shutter that successively activates each pixel column and its imaging plane one at a time from left to right. The time interval between the activation of two adjacent pixel columns is determined by the pixel clock.

3.3 Working principle of light curtains

The latest version of light curtains [BWWN19] works by rapidly rotating a light sheet laser using a galvo-mirror in synchrony with the motion of a vertically aligned camera's rolling shutter. A laser beam is collimated and shaped into a line sheet using appropriate lenses and is reflected at a desired angle using a controllable galvanometer mirror (see Fig. 3.1(b)). The illumination plane created by the laser intersects the active imaging plane of the camera in a vertical line along the curtain profile (Fig. 3.1(a)). Object points intersecting the vertical line of the ruled surface are imaged brightly in the corresponding camera column. The xz-projection (or "top-down" projection) of this vertical line intersecting the t-th imaging plane lies on R_t, and we call this the t-th "control point", denoted by $\mathbf{X}_t$ (Fig. 3.1(b)).

Light curtains have thickness: Although the illustrations in Fig. 3.1 (a, b) suggest that a light curtain is a 2D surface, in practice it is a 3D planar slab with a thickness. This is due to two reasons. First, the camera's pixels are not points but have a finite area. Therefore, the imaging plane is a *pyramid* originating from the camera's aperture. Second, the laser line sheet is not perfectly collimated; the laser beam diverges as it travels. This results in the illumination plane also being pyramidal. The intersection of the two pyramidal volumes results in a finite 3D volume; a visible object surface that lies anywhere in the intersection volume is imaged by the active camera column. Therefore, light curtains need not coincide exactly with the location of the visible object surface; the latter can be imaged if it is within the the curtain's thickness.

3.4 Light curtain input

The shape of a light curtain is uniquely defined by where it intersects each camera ray in the xz-plane, i.e. the control points $\{\mathbf{X}_1, \ldots, \mathbf{X}_T\}$. These will act as inputs to the light curtain device. In order to produce the light curtain defined by $\{\mathbf{X}_t\}_{t=1}^T$, the galvanometer is programmed to compute and rotate at, for each camera ray R_t, the reflection angle $\theta_t(\mathbf{X}_t)$ of the laser beam such that the laser sheet intersects R_t at $\mathbf{X}_t$. By selecting a control point on each camera ray, the light curtain device can be made to image any vertical ruled surface [BWWN19, WBW+18].

3.5 Light curtain output

The light curtain outputs an intensity value for each camera pixel.

- We can convert the light curtain image to a point cloud of all 3D visible points in the scene that intersect the light curtain surface. The density of light curtain points on the surface is usually much higher than LiDAR points. Chapter 5 uses this form of output.
- Alternatively, since a light curtain profile is specified by a control point $\mathbf{X}_t$ for every camera ray R_t in the top-down view, we compute the maximum pixel intensity value $\mathbf{I}_t$ of the t-th pixel column and treat this as the output of the light curtain for the corresponding ray R_t. Chapter 7 uses this form of output.

3.6 Light curtain constraints

The set of realizable curtains depend on the physical constraints imposed by the real device. The rotating galvo-mirror can operate at a maximum angular velocity $\omega_{\max}$ and a maximum angular acceleration $\alpha_{\max}$. Let $\mathbf{X}_{t-1}, \mathbf{X}_t, \mathbf{X}_{t+1}$ be the control points imaged by the light curtain on three consecutive camera rays. These induce laser angles $\theta(\mathbf{X}_{t-1}), \theta(\mathbf{X}_t), \theta(\mathbf{X}_{t+1})$ respectively. Let Δt be the time difference between when any two consecutive camera pixel columns are actived. Let $\mathbf{\Omega}_t = (\theta(\mathbf{X}_t) - \theta(\mathbf{X}_{t-1}))/\Delta t$ be the angular velocity of the galvo-mirror at R_t. Its angular acceleration at R_t is $(\mathbf{\Omega}_{t+1} - \mathbf{\Omega}_t)/\Delta t = (\theta(\mathbf{X}_{t+1}) + \theta(\mathbf{X}_{t-1}) - 2 \cdot \theta(\mathbf{X}_t))/(\Delta t)^2$. Then, the light curtain velocity and acceleration constraints, in terms of the control points $\mathbf{X}_t$, are:

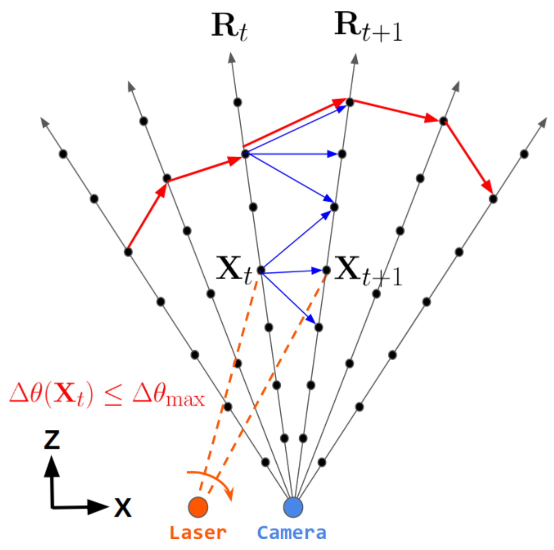
$$|\theta(\mathbf{X}_t) - \theta(\mathbf{X}_{t-1})| \leq \omega_{\max} \cdot \Delta t \qquad 2 \leq t \leq T \tag{3.1}$$

$$|\theta(\mathbf{X}_{t+1}) + \theta(\mathbf{X}_{t-1}) - 2\theta(\mathbf{X}_t)| \leq \alpha_{\max} \cdot (\Delta t)^2 \qquad 2 \leq t < T \tag{3.2}$$

CHAPTER 4

Constraint Graph: Feasible Curtains, Optimization & Probabilistic Analysis

4.1 Light curtain constraint graph



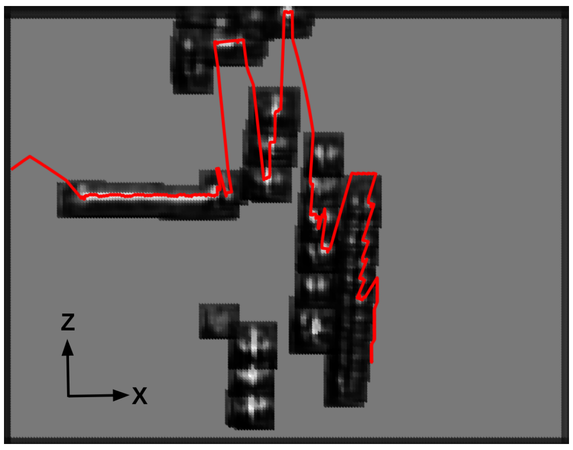


Figure 4.1: (a) Light curtain constraint graph. Black dots are nodes and blue arrows are the edges of the graph. The optimized light curtain profile is depicted as red arrows. (b) Example uncertainty map from the detector and optimized light curtain profile in red. Black is lowest uncertainty and white is highest uncertainty. The optimized light curtain covers the most uncertain regions.

We encode the light curtain constraints described in Section 3.6 into a graph, as illustrated in Figure 4.1. Each black ray corresponds to a camera ray. Each black dot on the ray is a vertex in the constraint graph. It represents a candidate control point and can be associated with any score that one may wish to optimize. Exactly one control point must be chosen per

We have open-sourced our planning and optimization library implementing the light curtain constraint graph at this repository: https://github.com/CMU-Light-Curtains/ConstraintGraph.

camera ray. The optimization objective could be to choose points in such a way that they maximize the total sum of scores. An edge between two control points indicates that the light curtain is able to transition from one control point $\mathbf{X}_t$ to the next, $\mathbf{X}_{t+1}$ without violating the *maximum velocity light curtain constraints*. Thus, the maximum velocity constraint (Eqn. 6.5) can be specified by restricting the set of edges (depicted using blue arrows). We note that the graph only needs to be constructed once and can be done offline.

4.2 Optimal light curtain placement

In this section, we will describe an exact optimization algorithm to maximize the objective function $J(\mathbf{X}_1, \ldots, \mathbf{X}_T) = \sum_{t=1}^{T} H(\mathbf{X}_t)$.

4.2.1 Constrained optimization

The control points $\{\mathbf{X}_t\}_{t=1}^T$, where each $\mathbf{X}_t$ lies on the the camera ray R_t, must be chosen to satisfy the physical constraints of the light curtain device: $|\theta(\mathbf{X}_{t+1}) - \theta(\mathbf{X}_t)| \leq \Delta\theta_{\text{max}}$ (see Sec. **??**: light curtain constraints). Hence, this is a constrained optimization problem. We discretize the problem by considering a dense set of N discrete, equally spaced points $\mathcal{D}_t = \{\mathbf{X}_t^{(n)}\}_{n=1}^N$ on each ray R_t. We will assume that $\mathbf{X}_t \in \mathcal{D}_t$ for all $1 \leq t \leq T$ henceforth unless stated otherwise. We use $N = 80$ in all our experiments which we found to be sufficiently large. Overall, the optimization problem can be formulated as:

$$\arg\max_{\{\mathbf{X}_t\}_{t=1}^T} \sum_{t=1}^{T} H(\mathbf{X}_t) \tag{4.1}$$

$$\text{where } \mathbf{X}_t \in \mathcal{D}_t \ \forall 1 \leq t \leq T \tag{4.2}$$

$$\text{subject to } |\theta(\mathbf{X}_{t+1}) - \theta(\mathbf{X}_t)| \leq \Delta\theta_{\text{max}}, \ \forall 1 \leq t < T \tag{4.3}$$

4.2.2 Dynamic programming for constrained optimization

The number of possible light curtain placements, $|\mathcal{D}_1 \times \cdots \times \mathcal{D}_T| = N^T$, is exponentially large, which prevents us from searching for the optimal solution by brute force. However, we observe that the problem can be decomposed into simpler subproblems. In particular, let us define $J_t^*(\mathbf{X}_t)$ as the optimal sum of uncertainties of the *tail subproblem* starting from $\mathbf{X}_t$ i.e.

$$J_t^*(\mathbf{X}_t) = \max_{\mathbf{X}_{t+1}, \ldots, \mathbf{X}_T} H(\mathbf{X}_t) + \sum_{k=t+1}^{T} H(\mathbf{X}_k); \tag{4.4}$$

$$\text{subject to } |\theta(\mathbf{X}_{k+1}) - \theta(\mathbf{X}_k)| \leq \Delta\theta_{\text{max}}, \ \forall\, t \leq k < T \tag{4.5}$$

If we were able to compute $J_t^*(\mathbf{X}_t)$, then this would help in solving a more complex subproblem using recursion: we observe that $J_t^*(\mathbf{X}_t)$ has the property of *optimal substructure,* i.e. the optimal solution of $J_{t-1}^*(\mathbf{X}_{t-1})$ can be computed from the optimal solution of $J_t^*(\mathbf{X}_t)$ via

$$\begin{aligned} J_{t-1}^*(\mathbf{X}_{t-1}) = H(\mathbf{X}_{t-1}) + &\max_{\mathbf{X}_t \in \mathcal{D}_t} J_t^*(\mathbf{X}_t) \\ &\text{subject to } |\theta(\mathbf{X}_t) - \theta(\mathbf{X}_{t-1})| \leq \Delta\theta_{\text{max}} \end{aligned} \tag{4.6}$$

Because of this optimal substructure property, we can solve for $J_{t-1}^*(\mathbf{X}_{t-1})$ via dynamic programming. We also note that the solution to $\max_{\mathbf{X}_1} J_1^*(\mathbf{X}_1)$ is the solution to our original constrained optimization problem (Eqn. 1-3).

We thus perform the dynamic programming optimization as follows: the recursion from Eqn. 4.6 can be implemented by first performing a backwards pass, starting from T and computing $J_t^*(\mathbf{X}_t)$ for each $\mathbf{X}_t$. Computing each $J_t^*(\mathbf{X}_t)$ takes only $O(B_{\text{avg}})$ time where B_{avg} is the average degree of a vertex (number of edges starting from a vertex) in the constraint graph, since we iterate once over all edges of $\mathbf{X}_t$ in Eqn. 4.6. Then, we do a forward pass, starting with $\arg\max_{\mathbf{X}_1 \in \mathcal{D}_1} J_1^*(\mathbf{X}_1)$ and for a given $\mathbf{X}_{t-1}^*$, choosing $\mathbf{X}_t^*$ according to Eqn. 4.6. Since there are N vertices per ray and T rays in the graph, the overall algorithm takes $O(NTB_{\text{avg}})$ time; this is a significant reduction from the $O(N^T)$ brute-force solution.

4.2.3 Hierarchical optimization objective for smoothness

If two light curtain placements produce the same sum of uncertainties, which one should we prefer? We propose a hierarchical optimization objective that prefers smoother light curtains. We show that this also has the optimal substructure property and can be optimized in a very similar manner.

Section 4.2 described an efficient algorithm for optimally placing light curtains to maximize coverage of high uncertainy regions. However, if two valid light curtain placements $\{\mathbf{X}'_t\}_{t=1}^T, \{\mathbf{X}''_t\}_{t=1}^T$ have equal sum of uncertainties, which one should we prefer? Distinct light curtain placements can have equal sums of uncertainties due to regions where the detector uncertainty is uniform. In such cases, we can choose to prefer curtains that are *smooth*, i.e. the laser angle has to change the least on average. We define a hierarchical objective function that ranks two placements as follows:

$$J_H(\{\mathbf{X}'_t\}_{t=1}^T) \geq J_H(\{\mathbf{X}''_t\}_{t=1}^T) \text{ iff } \begin{cases} J(\{\mathbf{X}'_t\}_{t=1}^T) > J(\{\mathbf{X}''_t\}_{t=1}^T) \\ \text{or} \\ \begin{cases} J(\{\mathbf{X}'_t\}_{t=1}^T) = J(\{\mathbf{X}''_t\}_{t=1}^T) \\ \qquad\qquad \text{and} \\ \sum_{t=1}^{T-1} |\theta(\mathbf{X}'_{t+1}) - \theta(\mathbf{X}'_t)|^2 \\ \leq \sum_{t=1}^{T-1} |\theta(\mathbf{X}''_{t+1}) - \theta(\mathbf{X}''_t)|^2 \end{cases} \end{cases}$$

This hierarchical objective prefers light curtains that cover a higher sum of uncertainties. But if two curtains have the same sum, this objective prefers the one with a lower sum of squared laser angle deviations. We note that this hierarchical objective $J_H(\mathbf{X}_1, \ldots, \mathbf{X}_T)$ also satisfies optimal substructure. In fact, it obeys the same recursive optimality equation as Equation 4.6. Hence, it can be accommodated by our approach with minimal modification to our algorithm. Additionally, it can be executed with no additional overhead in $O(NTB_{\text{avg}})$ time, and leads to smoother light curtains.

4.3 Extended constraint graph

The light curtain constraint graph, as described in Section 4.1 computes feasible light curtains that only satisfy the velocity constraints of the light curtain. Let us denoted this by $\mathcal{G}$. $\mathcal{G}$ has two components: a set of nodes $\mathbf{N}_t$ associated with each camera ray and edges between nodes $\mathbf{N}_t$ and $\mathbf{N}_{t+1}$ on consecutive camera rays. Nodes are designed to store information that fully captures the *state* of the galvo-mirror when imaging the ray R_t. An edge exists from $\mathbf{N}_t$ to $\mathbf{N}_{t+1}$ *iff* the galvo-mirror is able to transition from the state defined by $\mathbf{N}_t$ to the state defined by $\mathbf{N}_{t+1}$ without violating any of velocity constraints (Eqn. (3.1)).

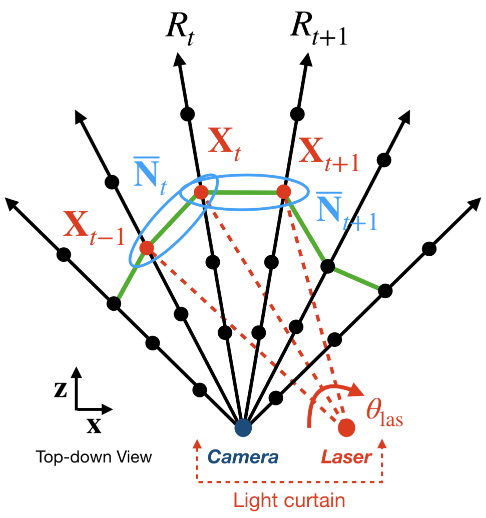


Figure 4.2: *Extended constraint graph:* light curtain constraint graph with our proposed extension. Black dots are control points that can be imaged, and blue ovals are extended nodes that contain two control points. Any path in this graph (shown in green) is a valid light curtain that satisfies velocity and acceleration constraints.

In Section 4.1, we defined the state to be $\mathbf{N}_t = \mathbf{X}_t$ (i.e. contain only one control point). But this representation does not ensure that acceleration constraints of Eqn. 3.2, that depend on three consecutive control points, are satisfied. This can produce light curtain profiles which require the galvo-mirror to change its angular velocity more abruptly than its physical torque limits can allow, resulting in hardware errors. Thus, we extend the definition of a node $\mathbf{N}_t$ at t to store control points on the current ray *and* the previous ray, as $\overline{\mathbf{N}}_t = (\mathbf{X}_{t-1}, \mathbf{X}_t)$ (see Fig. 4.2 (c)). Intuitively, $\overline{\mathbf{N}}$ contains information about the angular position and velocity of the galvo-mirror. This allows us to incorporate acceleration constraints by creating an edge between nodes $(\mathbf{X}_{t-1}, \mathbf{X}_t)$ and $(\mathbf{X}_t, \mathbf{X}_{t+1})$, *iff* $\mathbf{X}_{t-1}, \mathbf{X}_t, \mathbf{X}_{t+1}$ satisfy the velocity and acceleration constraints defined in Equations (3.1, 3.2). Thus, any path in the extended graph $\overline{\mathcal{G}}$ represents a feasible light curtain that satisfies both constraints. The acceleration constraints also serve to limit the increase in the number of nodes with feasible edges, keeping the graph size manageable.

4.4 Random curtains & theoretical guarantees

In this section, we show how a "random curtain"', i.e. a feasible curtain that is placed at random locations in the scene, can be sampled from the extended constraint graph $\overline{\mathcal{G}}$. We also show how to analytically compute the probability of a random curtain detecting an obstacle, which helps to probabilistically guarantee obstacle detection of our overall method.

4.4.1 Sampling random curtains from the constraint graph

First, we need to define a probability distribution over the set of valid curtains in order to sample from it. We do so by defining, for each node $\overline{\mathbf{N}}_t = (\mathbf{X}_{t-1}, \mathbf{X}_t)$, a *transition probability distribution* $P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$. This denotes the probability of transitioning from imaging the control points $\mathbf{X}_{t-1}, \mathbf{X}_t$ on the previous and current camera rays to the control point $\mathbf{X}_{t+1}$ on the next ray. We constrain $P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$ to equal 0 if there is no edge from node $(\mathbf{X}_{t-1}, \mathbf{X}_t)$ to node $(\mathbf{X}_t, \mathbf{X}_{t+1})$; an edge will exist *iff* the transition

$\mathbf{X}_{t-1} \to \mathbf{X}_t \to \mathbf{X}_{t+1}$ satisfies the light curtain constraints. Thus, $P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$ defines a probability distribution over the neighbors of $\overline{\mathbf{N}}_t$ in the constraint graph.

The transition probability distribution enables an algorithm to sequentially generate a random curtain. We begin by sampling the control points $\overline{\mathbf{N}}_2 = (\mathbf{X}_1, \mathbf{X}_2)$ according to an initial probability distribution $P(\overline{\mathbf{N}}_2)$. At the t-th iteration, we sample $\mathbf{X}_{t+1}$ according to the transition probability distribution $\mathbf{X}_{t+1} \sim P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$ and add $\mathbf{X}_{t+1}$ to the current set of sampled control points. After $(T-1)$ iterations, this process generates a full random curtain. Pseudo-code for this process is found in Algorithm 1 in Appendix 4.4.3. Our random curtain sampling process provides the flexibility to design any initial and transition probability distribution. See Appendix 4.4.3 for a discussion on various choices of the transition probability distribution, where we also provide a theoretical and empirical justification to use one distribution in particular.

4.4.2 Theoretical guarantees for random curtains

In this section, we first describe a procedure to detect objects in a scene using the output of a random light curtain placement. Then, we develop a method that runs dynamic programming on $\overline{\mathcal{G}}$ to analytically compute a random curtain's probability of detecting a specific object. This provides probabilistic safety guarantees on how well a random curtain can discover the safety envelope of an object.

Detection using light curtains: Consider an object in the scene whose visible surface intersects each camera ray at the positions $O_{1:T}$. This representation captures the position, shape and size of the object from the top-down view. Let $\{\mathbf{X}_t\}_{t=1}^T$ be the set of control points for a light curtain placed in the scene. The light curtain will produce an intensity $\mathbf{I}_t(\mathbf{X}_t, O_t)$ at each control point $\mathbf{X}_t$ that is sampled by the light curtain device. Note that $\mathbf{I}_t$ is a function of the position of the object as well as the position of the light curtain; the intensity increases as the distance between $\mathbf{X}_t$ and O_t reduces and is the highest when $\mathbf{X}_t$ and O_t coincide. We say that an object has been detected at control point $\mathbf{X}_t$ if the intensity $\mathbf{I}_t$ is above a threshold τ; the intensity threshold is used to account for noise in the image. We define a binary detection variable to indicate whether a detection of an object occurred at position O_t at control point $\mathbf{X}_t$ as $\mathbf{d}_t(\mathbf{X}_t, O_t) = [\mathbf{I}_t(\mathbf{X}_t, O_t) > \tau]$, where $[\cdot]$ is the indicator function. We declare that an object has been detected by a light curtain if it is detected on any of its control points. Formally, we define a binary detection variable to indicate whether a detection of object $O_{1:T}$ occurred at any of its control point $\mathbf{X}_{1:T}$ as $\mathbf{d}(\mathbf{X}_{1:T}, O_{1:T}) = \bigvee_{t=1}^T \mathbf{d}_t(\mathbf{X}_t, O_t)$, (where $\bigvee$ is 'logical or' operator). Our objective is then to compute the *detection probability*, denoted as $P(\mathbf{d}(\mathbf{X}_{1:T}, O_{1:T}))$, which is the probability that a curtain sampled from $\overline{\mathcal{G}}$ (using the sampling procedure described in Sec. 4.4.1) will detect the object $O_{1:T}$; below we will use the simpler notation $P(\mathbf{d})$ to denote the detection probability.

Theoretical guarantees using dynamic programming: The simplest method to compute the detection probability for a given object is to sample a large number of random curtains and output the average number of curtains that detect the object. However, a large number of samples would be needed to provide accurate estimates of the detection probability; further, this procedure is stochastic and the probability estimate will only be approximate. Instead, we propose utilizing the known structure of the constraint graph and the transition probabilities; we will apply dynamic programming to compute the detection probability both efficiently and analytically.

Our analytic method for computing the detection probability proceeds as follows: we first compute the value of the detection event $\mathbf{d}_t(\mathbf{X}_t, O_t)$ at every possible control point $\mathbf{X}_t$ that is part of a node in the constraint graph $\overline{\mathcal{G}}$ i.e. we compute whether or not a curtain placed at $\mathbf{X}_t$ is able to detect the object. Given the positions $O_{1:T}$ of an object, as well as the physical properties of the light curtain device (intrinsics of the camera and the power, thickness and divergence of the laser beam), we use a light curtain simulator to compute $\mathbf{I}_t(\mathbf{X}_t, O_t)$ using standard raytracing and rendering, for any arbitrary control point $\mathbf{X}_t$.

To compute the detection probability $P(\mathbf{d})$, we first define the notion of a "sub-curtain," which is a subset of the control points $\mathbf{X}_{t:T}$ which start at ray R_t and ends on ray R_T. We can decompose the overall problem of computing $P(\mathbf{d})$ into simpler sub-problems by defining the *sub-curtain detection probability* $P_{\text{det}}(\mathbf{X}_{t-1}, \mathbf{X}_t)$. This is the probability that any random sub-curtain starting at $(\mathbf{X}_{t-1}, \mathbf{X}_t)$ and ending on the last camera ray R_T detects the object O at some point between rays R_t and R_T. Using this definition, we can write the sub-curtain detection probability as $P_{\text{det}}(\mathbf{X}_{t-1}, \mathbf{X}_t) = P(\bigvee_{t'=t}^{T} \mathbf{d}_{t'}(\mathbf{X}_{t'}, O_{t'}) \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$.

Note that the overall curtain detection probability can be written in terms of the sub-curtain detection probabilities of the second ray (the first set of nodes in the graph) as

$$P(\mathbf{d}) = \sum_{\mathbf{X}_1, \mathbf{X}_2} P_{\text{det}}(\mathbf{X}_1, \mathbf{X}_2)\ P(\mathbf{X}_1, \mathbf{X}_2). \tag{4.7}$$

This is the sum of the detection probabilities of a random curtain starting from the initial nodes $P_{\text{det}}(\mathbf{X}_1, \mathbf{X}_2)$, weighted by the probability of the nodes being sampled from the initial distribution $P(\mathbf{X}_1, \mathbf{X}_2)$. Conveniently, the sub-curtain detection probabilities satisfy a simple recursive equation:

$$P_{\text{det}}(\mathbf{X}_{t-1}, \mathbf{X}_t) = \begin{cases} 1 & \text{if } \mathbf{d}_t(\mathbf{X}_t, O_t) = 1 \\ \\ \sum\limits_{\mathbf{X}_{t+1}} P_{\text{det}}(\mathbf{X}_t, \mathbf{X}_{t+1})\ P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t) & \text{otherwise} \end{cases} \tag{4.8}$$

Intuitively, if the control point $\mathbf{X}_t$ is able to detect the object, then the sub-curtain detection probability is 1 regardless of how the sub-curtain is placed on the later rays. If not, then the detection probability should be equal to the sum of the sub-curtain detection probabilities of the nodes the curtain transitions to, weighted by the transition probabilities.

This recursive relationship can be exploited by successively computing the sub-curtain detection probabilities from the last ray to the first. The sub-curtain detection probabilities on the last ray will simply be either 1 or 0, based on whether the object is detected there or not. After having computed sub-curtain detection probabilities for all rays between $t+1$ and T, the probabilities for nodes at ray t can be computed using the above recursive formula (Eqn. 4.8). Finally, after obtaining the probabilities for nodes on the second ray, the overall curtain detection probability can be computed as described using Eqn. 4.7. Pseudocode for this method can be found in Algorithm 2 in Appendix 4.4.4. A discussion on the computational complexity of the extended constraint graph $\overline{\mathcal{G}}$ (which contains more nodes and edges than $\mathcal{G}$) can be found in Appendix 4.5.

We have created a web-based demo (available on the project website) that computes the probability of a random curtain detecting an object with a user-specified shape in the top-down view. It also performs analysis of the detection probability as a function of the number of light curtains placed. In Section 7.3.1, we use this method to analyze the detection probability of random curtains as a function of the object size and number of curtain placements. We compare it against a sampling-based approach and show that our method gives the same results but with an efficient analytical computation.

4.4.3 Transition distributions for sampling random curtains

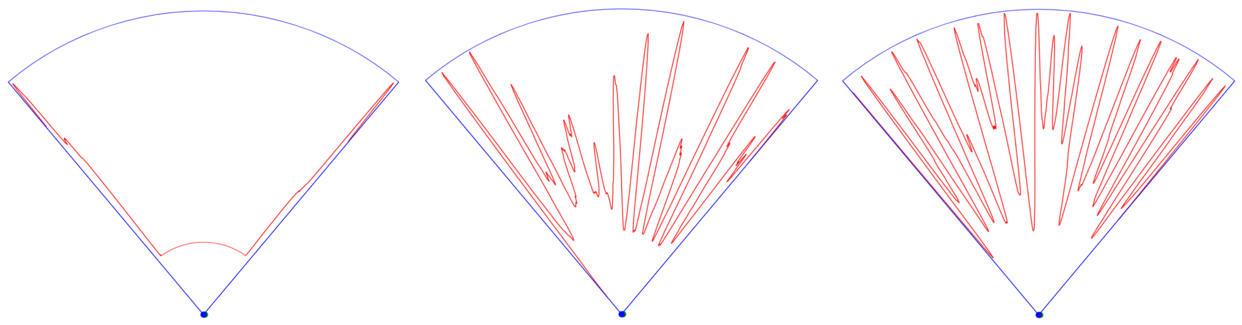
We first describe, in Algorithm 35 the procedure to sample a random curtain from the extended constraint graph $\overline{\mathcal{G}}$ by successively generating it using a transition probability function $P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$. The only constraint on $P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$ is that it must equal to 0 if $\mathbf{X}_{t-1}, \mathbf{X}_t, \mathbf{X}_{t+1}$ do not satisfy both the velocity and acceleration constraints of Equations (3.1, 3.2).

Algorithm 4.1: Sampling a random curtain from $\overline{\mathcal{G}}$

```
/* Inputs                                                   */
$\overline{\mathcal{G}} \leftarrow$ extended constraint graph
$P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t) \leftarrow$ transition prob. distribution
$P((\mathbf{X}_1, \mathbf{X}_2)) \leftarrow$ initial probability distribution

/* Progressively generate curtain                           */
Curtain $\leftarrow \{\}$

/* Initialization                                           */
Sample $(\mathbf{X}_1, \mathbf{X}_2) \sim P((\mathbf{X}_1, \mathbf{X}_2))$
Curtain $\leftarrow \{\mathbf{X}_1, \mathbf{X}_2\}$

/* Iteration                                                */
for $t = 2$ to $T - 1$ do
    $\mathbf{X}_{t+1} \sim P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$
    Curtain $\leftarrow$ Curtain $\cup \{\mathbf{X}_{t+1}\}$
end

return Curtain
```

We initialize by sampling a location $\overline{\mathbf{N}}_2 = (\mathbf{X}_1, \mathbf{X}_2)$ according to an initial sampling distribution. At the $(t-1)$-th iteration, we will have sampled the $t-1$ nodes $(\overline{\mathbf{N}}_2, \ldots, \overline{\mathbf{N}}_t)$ corresponding to the first t control points $(\mathbf{X}_1, \ldots, \mathbf{X}_t)$ of the random curtain. At the t-th iteration, we sample $\mathbf{X}_{t+1}$ according to the transition probability distribution $\mathbf{X}_{t+1} \sim P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$ and add $\mathbf{X}_{t+1}$ to the current set of control points. After all iterations are over, this algorithm generates a complete random curtain.

The above procedure to sample random curtain provides the flexibility to design any initial and transition probability distribution functions. Then, what are good candidate distributions? We use random curtains to detect the presence of objects in the scene whose location is unknown. Hence, the objective is to find the light curtian sampling distribution that maximizes the probability of detection of an object that might be placed at any arbitrary location in a scene. This objective will be achieved by random curtains that *cover a large area*. We now discuss a few sampling methods and qualitatively evaluate them in terms of the area covered by random curtains generated from them.



(a) Uniform sampling of neighbors (b) Uniform linear sampling of setpoints (c) Uniform area sampling of setpoints

Figure 4.3: Qualitative comparision of the coverage of random light curtains under different transition probability distributions. Sampled random curtains are shown in red. (a) *Uniform neighbor sampling*: for a given node, its neighbors on the next camera ray are sampled uniformly at random. This can produce random curtains that are at a constant distance away from the device. (b) *Uniform linear setpoint sampling*: for every camera ray, a setpoint distance $r \in [0, r_{\text{max}}]$ is sampled uniformly at random. Then the neighbor closest to the setpoint is chosen. This has significantly higher coverage, but is biased towards sampling locations close to the device. (c) *Uniform area setpoint sampling*: for every camera ray, a setpoint distance $r \in [0, r_{\text{max}}]$ is sampled with a probability proportional to r. This assigns a higher probability to a larger r, and corresponds to uniform *area* sampling. Then the neighbor closest to the setpoint is chosen. This method qualitatively exhibits the best coverage.

1. Uniform neighbor sampling: Perhaps the simplest transition probability distribution $P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$ for a given extended node $(\mathbf{X}_{t-1}, \mathbf{X}_t)$ is to select a neighboring control point $\mathbf{X}_{t+1}$ (that is connected by a valid edge originating from the node) with uniform probability. However, since the distribution does not take into account the physical locations of the current control point, it does not explicitly try to maximize coverage. To illustrate this, consider a random curtain that starts close to light curtain device. If it were to maximize coverage, the galvanometer would need to rotate so that the light curtain is placed farther from the device on subsequent camera rays. However, since its neighboring nodes are selected at random, the sampled locations on the next ray are equally likely to be nearer to the device than farther away from it. This can produce random curtains as shown in Fig. 4.3 (a).

2. Uniform linear setpoint sampling: a more principled way to sample neighbors is inspired by rapidly-exploring random trees (RRTs), which are designed to quickly explore and cover a given space. During tree expansion, an RRT first randomly samples a *setpoint* location, and selects the vertex that is closest to that location. We adopt a similar procedure. For any current node $(\mathbf{X}_{t-1}, \mathbf{X}_t)$, we first sample a setpoint distance $r \in [0, r_{\text{max}}]$ uniformly at random on a line along the camera ray. The probability density of r is a constant and equal to $P(r) = 1/r_{\text{max}}$. Then, we select a valid neighbor $\mathbf{X}_{t+1}$ that is closest to this setpoint location among all valid neighbors. Let us again consider the situation described in the previous approach, where the current node is located close to the light curtain device. When a setpoint is sampled uniformly along the next camera ray, there is high probability that it will correspond to a location that is farther away from the current node. Hence, the neighboring location $\mathbf{X}_{t+1}$ that is chosen on the next ray will likely lie away from the device as well. A random curtain sample generated from this distribution is shown in Fig. 4.3 (b). It tends to alternate between traversing near regions and far regions of the space in front of the curtain, covering a larger area than the previous sampling approach.

3. Uniform area setpoint sampling: We use the setpoint sampling approach described above, but revisit how the setpoint itself is sampled. Previously, the setpoint $r \in [0, r_{\max}]$ was sampled uniformly at random on the line along the ray, with $P(r) = 1/r_{\max}$. Now, we propose an alternate sampling distribution and provide some theoretical justification. Since we want to uniformly cover the area in front of the light curtain device, consider the following experiment. Let us sample a point (x, y) uniformly from the area within a circle of radius $r_{\max}$ (or equivalently, within any sector of that circle). Then the cumulative distribution function of $r = \sqrt{x^2 + y^2}$ is $P(r < r') = \pi r'^2 / \pi r_{\max}^2$ (area of the smaller circle divided by the area of entire circle), which implies that the probability density function of r is equal to $P(r) = 2r/r_{\max}^2$. This suggests that we must assign a higher probability to a larger r (proportional to r), since a larger area exists away from the center of the circle than near the center. Hence, we sample the setpoint r from $P(r) = 2r/r_{\max}^2$, by first sampling $s \sim$ Uniform$(0, r_{\max}^2)$ and then setting $r = \sqrt{s}$. Finally, we select the valid neighbor $\mathbf{X}_{t+1}$ on the next ray that is closest to this setpoint. Since this method is motivated by sampling areas rather than sampling along a line, we call this approach "area setpoint sampling". An example curtain sampled using this approach is visualized in Fig. 4.3 (c), which generally exhibits the best coverage among all methods. We use this method to sample random light curtains for all experiments in this paper.

4.4.4 Dynamic programming for computing detection probability

To compute the quantity $P(\mathbf{d})$, we first decompose the overall problem into smaller subproblems by defining the *sub-curtain detection probability* $P_{\text{det}}(\mathbf{X}_{t-1}, \mathbf{X}_t) = P(\vee_{t'=t}^{T} \mathbf{d}_t \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$. This is the probability that a random curtain *starting* on the extended node $(\mathbf{X}_{t-1}, \mathbf{X}_t)$ and ending on the last camera ray, detects the object O between rays R_t and R_T. Note that the overall detection probability can be written in terms of the sub-curtain detection probabilities of the second ray as $P(\mathbf{d}) = \sum_{\mathcal{S}_2} P(\mathbf{X}_1, \mathbf{X}_2) \cdot P_{\text{det}}(\mathbf{X}_1, \mathbf{X}_2)$. Then we iterate over camera rays from R_T to R_2. The node detection probabilities on the last ray will simply be either 1 or 0, based on whether the object is detected at the node or not. After having computed node detection probabilities for all rays between $t+1$ and T, the probabilities for nodes at ray t can be computed using a recursive formula. Finally, after obtaining the probabilities for nodes on the initial rays, the overall detection probabilities can be computed as described previously.

4.5 Computational complexity of the extended constraint graph

In this section, we discuss the computational complexity associated with the extended constraint graph. Let K be the number of discretized control points per camera ray in the constraint graph, and let T be the number of camera rays.

Constraint graph size: in the original constraint graph $\mathcal{G}$ of Ancha et. al. [ARH+20], since a node $\mathbf{N}_t = \mathbf{X}_t$ contains only one control point, there can be $O(K)$ nodes per camera ray and $O(K^2)$ edges between consecutive camera rays. This means that there are $O(TK)$ nodes and $O(TK^2)$ edges in the graph.

Algorithm 4.2: Dynamic programming to compute detection probabilities $\overline{\mathcal{G}}$

```
/* Inputs                                                              */
$\overline{\mathcal{G}} \leftarrow$ extended constraint graph
$P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t) \leftarrow$ transition prob. distribution
$P((\mathbf{X}_1, \mathbf{X}_2)) \leftarrow$ initial probability distribution
$\{O_1, \ldots, O_T\} \leftarrow$ ground truth object locations
$\tau \leftarrow$ detection intensity threshold

$\mathcal{S}_t \leftarrow$ the set of nodes on the $t$-th camera ray in the constraint graph.

/* Detection at each location                                          */
forall $\mathbf{X}_t \in \mathcal{S}_t,\ \ 1 \leq t \leq T$ do
    $\mathbf{I}_t(\mathbf{X}_t \mid O_t) \leftarrow$ intensity using rendering
    $\mathbf{d}_t(\mathbf{X}_t \mid O_t) \leftarrow [\mathbf{I}_t(\mathbf{X}_t \mid O_t) > \tau]$
end

/* Initializing last ray                                               */
for $(\mathbf{X}_{T-1}, \mathbf{X}_T) \in \mathcal{S}_T$ do
    $P_{\text{det}}(\mathbf{X}_{T-1}, \mathbf{X}_T) \leftarrow \mathbf{d}_t(\mathbf{X}_T \mid O_T)$
end

/* Dynamic programming loop                                            */
for $t = T - 1$ to $2$ do
    for $(\mathbf{X}_{T-1}, \mathbf{X}_T) \in \mathcal{S}_t$ do
        if $\mathbf{d}_t(\mathbf{X}_t, O_t) = 1$ then
            $P_{\text{det}}(\mathbf{X}_{T-1}, \mathbf{X}_T) \leftarrow 1$
        end
        else
            $P_{\text{det}}(\mathbf{X}_{T-1}, \mathbf{X}_T) \leftarrow \sum_{\mathbf{X}_{T+1}} P_{\text{det}}(\mathbf{X}_T, \mathbf{X}_{T+1}) \cdot P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$
        end
    end
end

/* Initial ray                                                         */
$P(\mathbf{d}) \leftarrow 0$
for $(\mathbf{X}_1, \mathbf{X}_2)$ do
    $P(\mathbf{d}) \leftarrow P(\mathbf{d}) + P(\mathbf{X}_1, \mathbf{X}_2) \cdot$
              $P_{\text{det}}(\mathbf{X}_1, \mathbf{X}_2)$
end

return $P(\mathbf{d})$
```

However, in the extended constraint graph $\overline{\mathcal{G}}$, each node $\overline{\mathbf{N}}_t = (\mathbf{X}_{t-1}, \mathbf{X}_t)$ contains a pair of control points. Hence, there can be up to $O(K^2)$ nodes per camera ray and $O(K^4)$ edges between consecutive camera rays! This implies that the total nodes and edges in the graph can be up to $O(TK^2)$ and $O(TK^4)$ respectively.

Dynamic programming: dynamic programming involves visiting each node and each edge in the graph once. Therefore, the worst-case computation time of dynamic programming in the extended constraint graph, namely $O(TK^4)$, might seem prohibitively large at first. However, the additional acceleration constraints in $\overline{\mathcal{G}}$ can significantly limit the increase in the number of nodes and edges. Additionally, we perform graph pruning as a post-processing step, to remove all nodes in the graph that do not have any edges. Since the topology of the constraint graph is fixed, the graph creation and pruning steps can be done offline and only once. These optimizations enable our dynamic programming procedure to be very efficient, as shown in Sec. 7.3.1. That being said, any slow down in dynamic programming is generally acceptable because it is only used for offline probabilistic analysis.

Random curtain generation: random curtains are placed by our online method. Fortunately, generating random curtains from the constraint graph is very fast. It involves a single forward pass (random walk) through the graph, visiting exactly one node per ray. It also involves parsing each visited node's transition probability distribution vector, whose length is equal to the number of edges of that node. Since both $\mathcal{G}$ and $\overline{\mathcal{G}}$ can have at most K edges per node, the runtime of generating a random curtain is $O(TK)$ (for both $\mathcal{G}$ and $\overline{\mathcal{G}}$). In practice, a large number of random curtains can be precomputed offline.

CHAPTER 5

Active Object Detection

5.1 Introduction

In this work, we propose a method for 3D object detection using programmable light curtains. The task of 3D object detection involves identify tight 3D bounding boxes around objects belonging to pre-defined semantic classes such as cars, pedestrians etc. We propose to use a deep neural network's prediction uncertainty as a guide for determining how to actively sense an environment. Our insight is that if a *controllable* sensor images the regions which the network is most uncertain about, the data obtained from those regions can help resolve the network's uncertainty and improve recognition. Conveniently, most deep learning based recognition systems output confidence maps, which can be used for this purpose when converted to an appropriate notion of uncertainty.

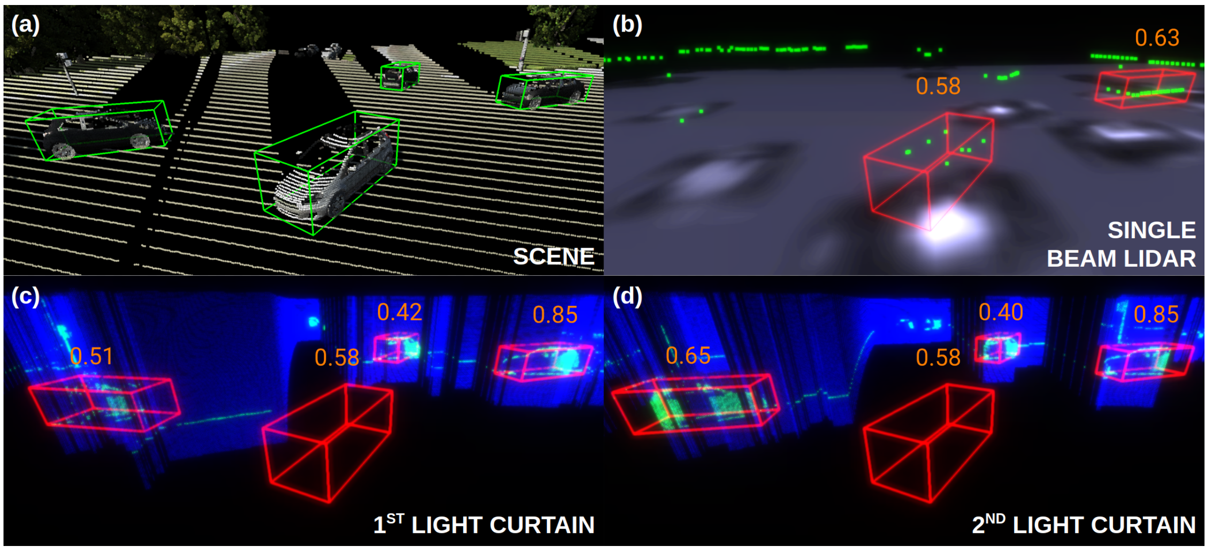


Figure 5.1: *Object detection using light curtains.* (a) Scene with 4 cars; ground-truth boxes shown in green. (b) Sparse green points are from a single-beam LiDAR; it can detect only two cars (red boxes). Numbers above detections boxes are confidence scores. Uncertainty map in greyscale is displayed underneath: whiter means higher uncertainty. (c) First light curtain (blue) is placed to optimally cover the most uncertain regions. Dense points (green) from light curtain results in detecting 2 more cars. (d) Second light curtain senses even more points and fixes the misalignment error in the leftmost detection.

Given neural network uncertainty estimates, we show how a light curtain can be placed to *optimally* cover the regions of maximum uncertainty. First, we use an information-gain based framework to propose placing light curtains that maximize the sum of uncertainties of the covered region (Sec. 5.2.3, Appendix 5.4). However, the structure of the light curtain and physical constraints of the device impose restrictions on how the light curtain can be placed. We pre-compute our previously defined "constraint graph", which describes all possible light curtain placements that respect these physical constraints. Using our optimization approach based on dynamic programming, we efficiently search over all possible feasible paths in the constraint graph and maximize this objective (Sec. 4.2). This is a novel approach to constrained optimization of a controllable sensor's trajectory which takes advantage of the properties of the problem we are trying to solve.

Our proposed active perception pipeline for 3D detection proceeds as follows. We initially record sparse data with an inexpensive single beam LiDAR sensor that performs fixed 3D scans. This data is input to a 3D point cloud object detector, which outputs an initial set of detections and confidence estimates. These confidence estimates are converted into uncertainty estimates, which are used by our dynamic programming algorithm to determine where to place the first light curtain. The output of the light curtain readings are again input to the 3D object detector to obtain refined detections and an updated uncertainty map. This process of estimating detections and placing new light curtains can be repeated multiple times (Fig. 7.7). Hence, we are able to sense the environment progressively, intelligently, and efficiently.

We evaluate our algorithm using two synthetic datasets of urban driving scenes [GWCV16, ZBGGV+19]. Our experiments demonstrate that our algorithm leads to a monotonic improvement in performance with successive light curtain placements. We compare our proposed optimal light curtain placement strategy to multiple baseline strategies and find that they are significantly outperformed by our method. To summarize, our contributions are the following:

- We propose a method for using a deep learning based 3D object detector's prediction uncertainty as a guide for active sensing (Sec. 5.2.2).
- Given a network's uncertainty, we derive an optimization objective to decide where to place light curtains using the principle of maximizing information gain (Sec. 5.2.3, Appendix 5.4).
- Our novel contribution is to encode the physical constraints of the device into a graph and use dynamic-programming based graph optimization to efficiently maximize the objective while satisfying the physical constraints (Sec. 5.2.3, 4.2).
- We show how to train such an active detector using online light curtain data generation (Sec. 5.2.4).
- We empirically demonstrate that our approach successively improves detection performance over LiDAR and is significantly better compared to a number of baseline approaches (Sec. 7.3).

The project webpage for this chapter can be found at:
https://siddancha.github.io/projects/active-perception-light-curtains/.
We have open sourced our code for active object detection using programmable light curtains at: https://github.com/CMU-Light-Curtains/ObjectDetection.

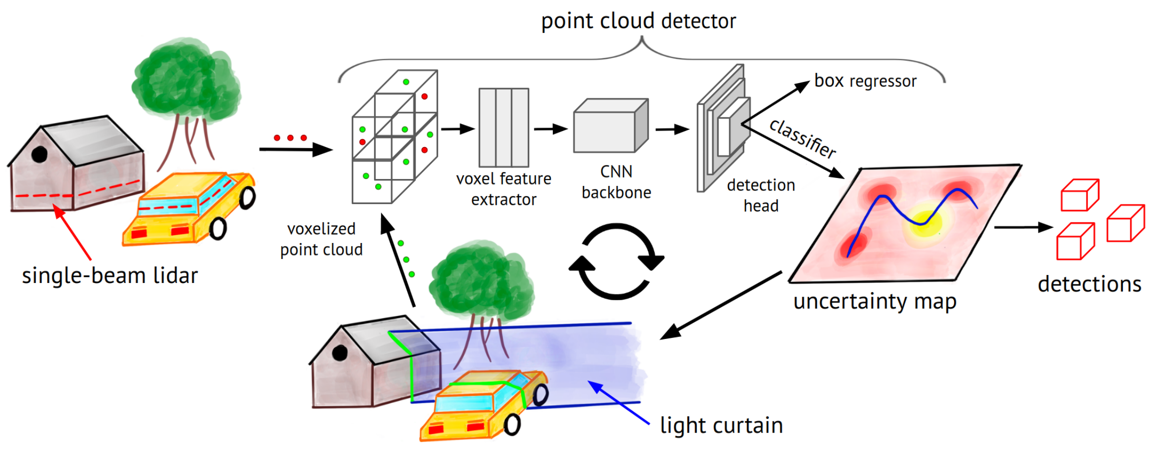


Figure 5.2: *Our method for detecting objects using light curtains.* An inexpensive single-beam lidar input is used by a 3D detection network to obtain rough initial estimates of object locations. The uncertainty of the detector is used to optimally place a light curtain that covers the most uncertain regions. The points detected by the light curtain (shown in green in the bottom figure) are input back into the detector so that it can update its predictions as well as uncertainty. The new uncertainty maps can again be used to place successive light curtains in an iterative manner, closing the loop.

5.2 Approach

5.2.1 Overview

Our aim is to use light curtains for detecting objects in a 3D scene. The overall approach is illustrated in Fig. 7.7. We use a voxel-based point cloud detector [YML18] and train it to use light curtain data without any architectural changes. The pipeline illustrated in Fig. 7.7 proceeds as follows.

To obtain an initial set of object detections, we use data from an inexpensive single-beam LiDAR as input to the detector. This produces rough estimates of object locations in the scene. Single-beam LiDAR is inexpensive because it consists of only one laser beam as opposed to 64 or 128 beams that are common in autonomous driving. The downside is that the data from the single beam contains very few points; this results in inaccurate detections and high uncertainty about object locations in the scene (see Fig. 5.1b).

Alongside bounding box detections, we can also extract from the detector an "uncertainty map" (explained in Sec. 5.2.2). We then use light curtains, placed in regions guided by the detector's uncertainty, to collect more data and iteratively refine the object detections. In order to get more data from the regions the detector is most uncertain about, we derive an information-gain based objective function that sums the uncertainties along the light curtain control points (Sec. 5.2.3 and Appendix 5.4), and we develop a constrained optimization algorithm that places the light curtain to maximize this objective (Sec. 4.2).

Once the light curtain is placed, it returns a dense set of points where the curtain intersects with visible objects in the scene. We maintain a *unified point cloud*, which we define as the union of all points observed so far. The unified point cloud is initialized with the points from the single-beam LiDAR. Points from the light curtain are added to the unified point cloud and this data is input back into the detector. Note that the input representation for the detector remains the same (point clouds), enabling the use of existing state-of-the-art point cloud detection methods without any architectural modifications.

As new data from the light curtains are added to the unified point cloud and input to the detector, the detector refines its predictions and improves its accuracy. Furthermore, the additional inputs cause the network to update its uncertainty map; the network may no longer be uncertain about the areas that were sensed by the light curtain. Our algorithm uses the new uncertainty map to generate a new light curtain placement. We can iteratively place light curtains to cover the current uncertain regions and input the sensed points back into the network, closing the loop and iteratively improving detection performance.

5.2.2 Extracting uncertainty from the detector

The standard pipeline for 3D object detection [ZT18, YML18, LVC+19] proceeds as follows. First, the ground plane (parallel to the xz-plane) is uniformly tiled with "anchor boxes"; these are reference boxes used by a 3D detector to produce detections. They are located on points in a uniformly discretized grid $G = [x_{\text{min}}, x_{\text{max}}] \times [z_{\text{min}}, z_{\text{max}}]$. For example, a $[-40\text{m}, 40\text{m}] \times [0\text{m}, 70.4\text{m}]$ grid is used for detecting cars in KITTI [GLSU13]. A 3D detector, which is usually a binary detector, takes a point cloud as input, and produces a binary classification score $p \in [0, 1]$ and bounding box regression offsets for every anchor box. The score p is the estimated probability that the anchor box contains an object of a specific class (such as car/pedestrian). The detector produces a detection for that anchor box if p exceeds a certain threshold. If so, the detector combines the fixed dimensions of the anchor box with its predicted regression offsets to output a detection box.

We can convert the confidence score to binary entropy $H(p) \in [0, 1]$ where $H(p) = -p \log_2 p - (1 - p) \log_2(1 - p)$. Entropy is a measure of the detector's uncertainty about the presence of an object at the anchor location. Since we have an uncertainty score at uniformly spaced anchor locations parallel to the xz-plane, they form an "uncertainty map" in the top-down view. We use this uncertainty map to place light curtains.

5.2.3 Information gain objective

Based on the uncertainty estimates given by Sec. 5.2.2, our method determines how to place the light curtain to sense the most uncertain/ambiguous regions. It seems intuitive that sensing the locations of highest detector uncertainty can provide the largest amount of information from a single light curtain placement, towards improving detector accuracy. As discussed in Sec. **??**, a single light curtain placement is defined by a set of T control points $\{\mathbf{X}_t\}_{t=1}^{T}$. The light curtain will be placed to lie vertically on top of these control points. To define an optimization objective, we use the framework of information gain (commonly used in next-best view methods; see Sec. 2.4) along with some simplifying assumptions (see Appendix 5.4). We show that under these assumptions, placing a light curtain to maximize information gain (a mathematically defined information-theoretic quantity) is equivalent to maximizing the objective $J(\mathbf{X}_1, \ldots, \mathbf{X}_T) = \sum_{t=1}^{T} H(\mathbf{X}_t)$, where $H(\mathbf{X})$ is the binary entropy of the detector's confidence at the anchor location of $\mathbf{X}$. When the control point $\mathbf{X}$ does not exactly correspond to an anchor location, we impute $H(\mathbf{X})$ by nearest-neighbor interpolation from the uncertainty map. Please see Appendix 5.4 for a detailed derivation.

5.2.4 Training active detector with online training data generation

We now describe our approach to train 3D point cloud detectors with data from light curtains and single-beam lidar. At each training iteration t, we retrieve a scene S_t from the training dataset. To create the input point cloud, we choose to either use the single-beam LiDAR data or k light curtain placements ($1 \leq k \leq K$), each of them with equal probability. For generating the k-th light curtain data, we start with the single-beam LiDAR point cloud. Then we successively perform a forward pass through the detector network with the current weights to obtain an uncertainty map. We compute the optimal light curtain placement for this map, gather points returned from placing this curtain, and finally, fuse the points back into the input point cloud. This cycle is repeated k times to obtain the input point cloud to train on. Generating light curtain data in such an *online* fashion ensures that the input distribution doesn't diverge from the network weights during the course of training. See Appendix 5.5 for more algorithmic details and an ablation experiment that evaluates the importance of online training data generation.

5.3 Experiments

Datasets: To evaluate our algorithm, we need dense ground truth depth maps to simulate an arbitrary placement of a light curtain. However, standard autonomous driving datasets, such as KITTI [GLSU13] and nuScenes [CBL+19], contain only sparse LiDAR data, and hence the data is not suitable to accurately simulate a dense light curtain to evaluate our method. To circumvent this problem, we demonstrate our method on two synthetic datasets that provide dense ground truth depth maps, namely the Virtual KITTI [GWCV16] and SYNTHIA [ZBGGV+19] datasets. Virtual KITTI is a photo-realistic synthetic video dataset designed for video understanding tasks [GWCV16]. It contains 21,160 frames (10,630 unique depth maps) generated from five different virtual worlds in urban driving settings design to closely resemble five scenes in the KITTI dataset, under different camera poses and weather conditions. It provides ground truth depth maps and 3D bounding boxes. We use four scenes (ids: 0001, 0006, 0018, 0020) as our training set, and one scene (id: 0002) as our test set.

We also use the latest version of the SYNTHIA dataset [ZBGGV+19] designed for active learning purposes. It is a large dataset containing photo-realistic scenes from urban driving scenarios, and provides ground truth depth and 3D bounding box annotations. It contains 191 training scenes (~96K frames) and 97 test scenes (~45K frames).

Evaluation metrics: We evaluate using common 3D detection metrics: mean average precision (mAP) of 3D bounding boxes (denoted as 3D mAP) and of 2D boxes in the bird's eye view (denoted as BEV mAP). We also evaluate using two different IoU overlap thresholds of 0.5 and 0.7 between detection boxes and ground-truth boxes to be considered true positives.

Our experiments demonstrate the following: First, we show that our method for successive placement of light curtains improves detection performance; particularly, there is a significant increase between the performance of single-beam LiDAR and the performance after placing the first light curtain. We also compare our method to multiple ablations and alternative placement strategies that demonstrate that each component of our method is crucial to achieve good performance. Finally, we show that our method can generalize to many more light curtain placements at test time than the method was trained on. In the appendix, we perform further experiments that include evaluating the generalization of our method to noise in the light curtain data, an ablation experiment for training with online data generation (Sec. 5.2.4), and efficiency analysis.

5.3.1 Comparison with varying number of light curtains

We train our method using online training data generation simultaneously on data from single-beam LiDAR and one, two, and three light curtain placements. We perform this experiment for both the Virtual KITTI and SYNTHIA datasets. The accuracies on their tests sets are reported in Table 5.1.

	Virtual KITTI				**SYNTHIA**			
	3D mAP		BEV mAP		3D mAP		BEV mAP	
	0.5 IoU	0.7 IoU	0.5 IoU	0.7 IoU	0.5 IoU	0.7 IoU	0.5 IoU	0.7 IoU
Single Beam Lidar	39.91	15.49	40.77	36.54	60.49	47.73	60.69	51.22
Single Beam Lidar (separate model)	42.35	23.66	47.77	40.15	60.69	48.23	60.84	57.98
1 Light Curtain	58.01	35.29	58.51	47.05	68.79	55.99	68.97	59.63
2 Light Curtains	60.86	37.91	61.10	49.84	69.02	57.08	69.17	67.14
3 Light Curtains	**68.52**	**38.47**	**68.82**	**50.53**	**69.16**	**57.30**	**69.25**	**67.25**

Table 5.1: Performance of the detector trained with single-beam LiDAR and up to three light curtains. Performance improves with more light curtain placements, with a significant jump at the first light curtain placement.

Note that there is a significant and consistent increase in the accuracy between single-beam LiDAR performance and the first light curtain placement (row 1 and row 3). This shows that actively placing light curtains on the most uncertain regions can improve performance over a single-beam LiDAR that performs fixed scans. Furthermore, placing more light curtains consistently improves detection accuracy.

As an ablation experiment, we train a separate model only on single-beam LiDAR data (row 2), for the same number of training iterations. This is different from row 1 which was trained with both single beam LiDAR and light curtain data but evaluated using only data for a single beam LiDAR. Although training a model with only single-beam LiDAR data (row 2) improves performance over row 1, it is still significantly outperformed by our method which uses data from light curtain placements.

Noise simulations: In order to simulate noise in the real-world sensor, we perform experiments with added noise in the light curtain input. We demonstrate that the results are comparable to the noiseless case, indicating that our method is robust to noise and is likely to transfer well to the real world. Please see Appendix 5.6 for more details.

5.3.2 Comparison with alternative light curtain placement strategies

In our approach, light curtains are placed by maximizing the coverage of uncertain regions using a dynamic programming optimization. How does this compare to other strategies for light curtain placement? We experiment with several baselines:

1. *Random*: we place frontoparallel light curtains at a random z-distance from the sensor, ignoring the detector's uncertainty map.
2. *Fixed depth*: we place a frontoparallel light curtain at a fixed z-distance (15m, 30m, 45m) from the sensor, ignoring the detector's uncertainty map.

3. *Greedy optimization*: this baseline tries to evaluate the benefits of using a dynamic programming optimization. Here, we use the same light curtain constraints described in Section 4.2 (Figure 4.1(a)). We greedily select the next control point based on local uncertainty instead of optimizing for the future sum of uncertainties. Ties are broken by (a) choosing smaller laser angle changes, and (b) randomly.

4. *Frontoparallel + Uncertainty*: Our optimization process finds light curtains with flexible shapes. What if the shapes were constrained to make the optimization problem easier? If we restrict ourselves to frontoparallel curtains, we can place them at the z-distance of maximum uncertainty by simply summing the uncertainties for every fixed value of z.

The results on the Virtual KITTI and SYNTHIA datasets are shown in Table 5.2. Our method significantly and consistently outperforms all baselines. This empirically demonstrates the value of using dynamic programming for light curtain placement to improve object detection performance.

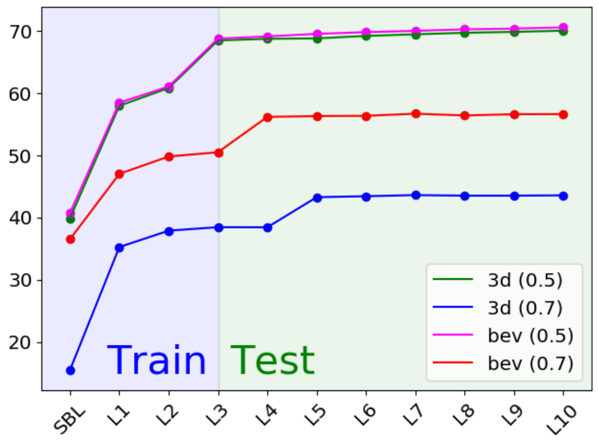
5.3.3 Generalization to successive light curtain placements

If we train a detector using our online light curtain data generation approach for k light curtains, can the performance generalize to more than k light curtains? Specifically, if we continue to place light curtains beyond the number trained for, will the accuracy continue improving? We test this hypothesis by evaluating on 10 light curtains, many more than the model was trained for (3 light curtains). Figure 5.3 shows the performance as a function of the number of light curtains. We find that in both Virtual KITTI and SYNTHIA, the accuracy monotonically improves with the number of curtains.

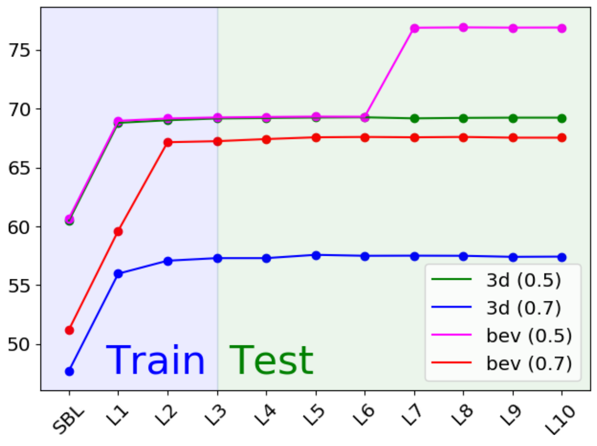
This result implies that a priori one need not worry about how many light curtains will be placed at test time. If we train on only 3 light curtains, we can place many more light curtains at test time; our results indicate that the performance will keep improving.

	Virtual KITTI				**SYNTHIA**			
	3D mAP		BEV mAP		3D mAP		BEV mAP	
	.5 IoU	.7 IoU	.5 IoU	.7 IoU	.5 IoU	.7 IoU	.5 IoU	.7 IoU
Random	41.29	17.49	46.65	38.09	60.43	47.09	60.66	58.14
Fixed depth - 15m	44.99	22.20	46.07	38.05	60.74	48.16	60.89	58.48
Fixed depth - 30m	39.72	19.05	45.21	35.83	60.02	47.88	60.23	57.89
Fixed depth - 45m	39.86	20.02	40.61	36.87	60.23	48.12	60.43	57.77
Greedy Optimization (Randomly break ties)	37.40	19.93	42.80	35.33	60.62	47.46	60.83	58.22
Greedy Optimization (Min laser angle change)	39.20	20.19	44.80	36.94	60.61	47.05	60.76	58.07
Frontoparallel + Uncertainty	39.41	21.25	45.10	37.80	60.36	47.20	60.52	58.00
Ours	**58.01**	**35.29**	**58.51**	**47.05**	**68.79**	**55.99**	**68.97**	**59.63**

Table 5.2: Baselines for alternate light curtain placement strategies, trained and tested on (a) Virtual KITTI and (b) SYNTHIA datasets. Our dynamic programming optimization approach significantly outperforms all other strategies.



(a) Generalization in Virtual KITTI



(b) Generalization in SYNTHIA

Figure 5.3: *Generalization to many more light curtains than what the detector was trained for.* We train using online data generation on single-beam lidar and only 3 light curtains. We then test with placing 10 curtains, on (a) Virtual KITTI, and (b) SYNTHIA. Performance continues to increase monotonically according to multiple metrics. Takeaway: one can safely place more light curtains at test time and expect to see sustained improvement in accuracy.

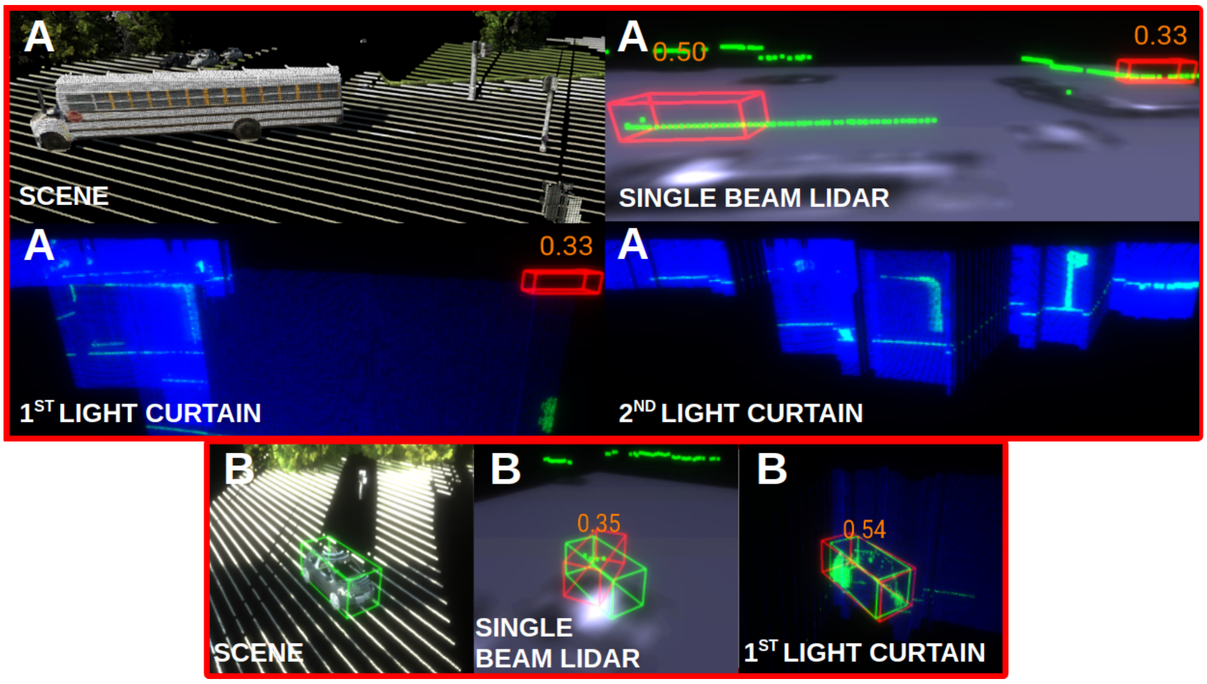


Figure 5.4: *Successful cases:* Other type of successful cases than Fig. 5.1. In (A), the single-beam LiDAR incorrectly detects a bus and a piece of lawn as false positives. They get eliminated successively after placing the first and second light curtains. In (B), the first light curtain fixes misalignment in the bounding box predicted by the single beam LiDAR.

5.3.4 Qualitative analysis

We visualized a successful case of our method in Fig. 5.1. This is an example where our method detects false negatives missed by the single-beam LiDAR. We also show two other types of successful cases where light curtains remove false positive detections and fix misalignment errors in Figure 5.4. In Figure 5.5, we show the predominant failure case of our method. See captions for more details.

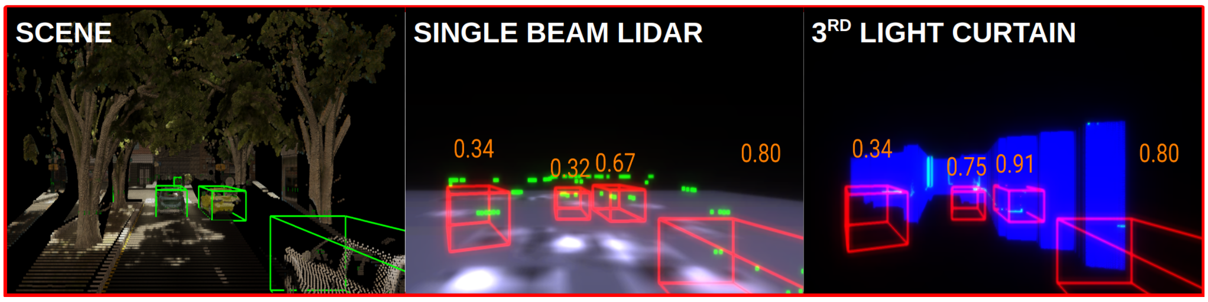


Figure 5.5: *Failure cases:* The predominant failure mode is that the single beam LiDAR detects a false positive which is not removed by light curtains because the detector is overly confident in its prediction (so the estimated uncertainty is low). *Middle*: Falsely detecting a tree to be a car. *Right*: After three light curtains, the detection persists because light curtains do not get placed on this false positive.

The predominant failure case of our method is when the LiDAR makes a mistake, such as a false positive in Fig. 5.5, but the light curtain fails to be placed in that region to fix the mistake. This happens when the detector makes a mistake but is very confident in its prediction; in such a case, the estimated uncertainty for this prediction will be low and a light curtain may not be placed at this location. In this particular example shown, after six light curtain placements, a light curtain eventually gets placed at the location of the false positive and the detector fixes its mistake. However, in other examples, a light curtain might never be placed at the location of the incorrect detection, due to an overly confident (but incorrect) prediction.

5.4 Information gain objective

In this section, we derive the optimization objective used in Sections 5.2.3 and 4.2, from a perspective of maximizing information gain. Information gain is a well-defined mathematical quantity, and choosing sensing actions to maximize information gain has been used as the basis of many works on next-best view planning (see Sec. **??**).

We will first describe some notation, and make two simplifying assumptions in order to derive our objective as an approximation of information gain.

5.4.1 Notation

- The detector predicts the probability of a detection at every anchor box location. Let there be a total of K discrete anchor box location, which are usually organized as a regular 2D-grid (see Sec. 5.2.2). Let $\mathbf{A}_k$ denote the k-th anchor box, where $1 \leq k \leq K$. Define $\mathbf{A} = \{\mathbf{A}_k\}_{k=1}^K$ to be the vector of all anchor boxes.

- Let $D_{\mathbf{A}_k}$ be a binary random variable denoting whether a detection exists at $\mathbf{A}_k$. $D_{\mathbf{A}_k} \in \{0, 1\}$; it is 0 if there is no detection at $\mathbf{A}_k$, and 1 if there is. Define $D_{\mathbf{A}} = \{D_{\mathbf{A}_k}\}_{k=1}^K$.

- Given a unified point cloud C, an inference algorithm (in this case, the detector) outputs a probability distribution $P(D_{\mathbf{A}} \mid C)$ over all possible detection states $D_{\mathbf{A}} \in \{0, 1\}^K$. Denote by $P(D_{\mathbf{A}_k})$ the marginal probability distribution of detection at $\mathbf{A}_k$.

- As discussed in Sec. **??**, a single light curtain placement is defined by a set of control points $L = \{\mathbf{X}_t\}_{t=1}^T$. The light curtain will be placed to lie vertically on top of these control points. The 3D points sensed by this light curtain are fused back into C, to obtain an updated unified point cloud C'. We assume for now that the control points $\mathbf{X}_t$ correspond to some anchor box locations.

5.4.2 Assumptions

We now make the following assumptions:

1. *Detections probabilities across locations are independent.*
   That is, $P(D_\mathbf{A} \mid C) = \prod_{k=1}^{K} P(D_{\mathbf{A}_k} \mid C)$. This is a reasonable assumption, since the probability of detections at one location should be unaffected by detections in other locations. A consequence of this assumption is that the overall entropy $H(D_\mathbf{A} \mid C)$ can be written as the sum of entropies over individual anchor locations i.e. $H(D_\mathbf{A} \mid C) = \sum_{k=1}^{K} H(D_{\mathbf{A}_k} \mid C)$ (since the entropy of independent random variables is the sum of their individual entropies).

2. *Light curtain sensing resolves uncertainty fully but locally.*
   After placing $L = \{\mathbf{X}_t\}_{t=1}^{T}$, updating the unified point cloud to C', re-running the detector, and obtaining a new probability distribution of the updated detections $D'_\mathbf{A}$, the following hold.

   a) The uncertainty of locations covered by the curtain reduces to zero:
   $P(D'_{\mathbf{A}_k} \mid C') \in \{0, 1\}$ for all $\mathbf{A}_k \in L$.

   b) The uncertainty of all the other locations remains unchanged:
   $P(D'_{\mathbf{A}_k} \mid C') = P(D_{\mathbf{A}_k} \mid C)$ for all $\mathbf{A}_k \notin L$.

Assumptions 1 and 2 imply that the entropy of the updated distribution is given by (here K is the total number of anchor locations, and T is the number of locations that the light curtain lies on).

$$\begin{aligned}
H(D'_\mathbf{A} \mid C') &= \sum_{k=1}^{K} H(D'_{\mathbf{A}_k} \mid C') \\
&= \sum_{\mathbf{A}_k \in L} \underbrace{H(D'_{\mathbf{A}_k} \mid C')}_{=\ 0 \text{ as } P(D'_{\mathbf{A}_k} \mid C') \in \{0,1\}} + \sum_{\mathbf{A}_k \notin L} \underbrace{H(D'_{\mathbf{A}_k} \mid C')}_{=\ H(D_{\mathbf{A}_k} \mid C)} \\
&= \sum_{\mathbf{A}_k} H(D_{\mathbf{A}_k} \mid C) - \sum_{\mathbf{A}_k \in L} H(D_{\mathbf{A}_k} \mid C) \\
&= \sum_{k=1}^{K} H(D_{\mathbf{A}_k} \mid C) - \sum_{\mathbf{A}_k \in L} H(D_{\mathbf{A}_k} \mid C) \\
&= H(D_\mathbf{A} \mid C) - \sum_{t=1}^{T} H(D_{\mathbf{X}_t} \mid C)
\end{aligned}$$

The information gain, which is essentially a difference between the prior and updated entropies, is

$$\begin{aligned}\text{Information Gain} &= H(D_{\mathbf{A}} \mid C) - H(D'_{\mathbf{A}} \mid C') \\ &= H(D_{\mathbf{A}} \mid C) - \Big(H(D_{\mathbf{A}} \mid C) - \sum_{t=1}^{T} H(D_{\mathbf{X}_t} \mid C)\Big) \\ &= \sum_{t=1}^{T} H(D_{\mathbf{X}_t} \mid C)\end{aligned}$$

Optimization objective: This leads us to an optimization objective where maximizing information gain is equivalent to simply maximizing the sum of uncertainties (binary entropies) over the control points the curtain lies on. The maximization objective then becomes: $J(\mathbf{X}_1, \ldots, \mathbf{X}_T) = \sum_{t=1}^{T} H(\mathbf{X}_t)$, where $H(\mathbf{X})$ is the binary entropy of the detector's confidence at the location of $\mathbf{X}$.

5.5 Training active detection with online light curtain data generation

In this section, we expand on the details of our method to train the detector described in Section 5.2.4. Note that we use the same detector to process data from the single beam LiDAR and all subsequent light curtain placements. During training, data instances need to be sampled from the single-beam LiDAR, as well as from up to K number of light curtain placements. We choose $K = 3$ in all our experiments. Crucially, since the light curtains are placed based on the output (uncertainty maps) of the detector, the point cloud data distribution from the k-th ($1 \leq k \leq K$) light curtain placement depends on the current weights of the detector. As the weights of the detector get updated during each gradient descent step, the input training data distribution from the k-th light curtain also changes. To accomodate for non-stationary training data, we propose *training with online data-generation*. This is described in Algorithm 21.

At each training iteration t, we retrieve a scene S_t from the training dataset. To create the input point cloud, we choose to either use the single-beam LiDAR data or k light curtain placements ($1 \leq k \leq K$), each of them with equal probability. For generating the k-th light curtain data, we start with the single-beam LiDAR point cloud. Then we successively perform a forward pass through the detector network with the current weights to obtain an uncertainty map. We compute the optimal light curtain placement for this map, gather points returned from placing this curtain, and finally, fuse the points back into the input point cloud. This cycle is repeated k times to obtain the input point cloud to train on. Generating light curtain data in such an *online* fashion ensures that the input distribution doesn't diverge from the network weights during the course of training.

Ablation experiment

Here, we perform an ablation experiment on the Virtual KITTI dataset, to evaluate the importance of training with online light curtain data generation. We first collect the entire dataset at the beginning, using the initial weights of the network. Then, we freeze this data and train the detector. The results are shown in Table 5.3. We see that the accuracy on light

Algorithm 5.1: Training with Online Light Curtain Data Generation

```
1  $W_0 \leftarrow$ initial weights of the detector
2  $T \leftarrow$ number of training iterations
3  $K \leftarrow$ number of light curtain placements
4  Function InputPointCloud($W$, $S$, $k$):
5      if $k = 0$ then
6          $P_0 \leftarrow$ point cloud from single-beam LiDAR in scene S
7          return $P_0$
8      else
9          $P_{k-1} \leftarrow$ InputPointCloud($W$, $S$, $k-1$)
10         $H \leftarrow$ uncertainty map from detector with weights $W$ and input $P_{k-1}$
11         $P \leftarrow$ point cloud from placing light curtain optimized for $H$ in scene $S$
12         $P_k \leftarrow P_{k-1} \cup P$
13         return $P_k$
14     end
15 for $t = 1$ to $T$ do
16     $S_t \leftarrow$ $t$-th training scene
17     $k_t \leftarrow$ randomly sample from $\{0, 1, \dots, K\}$
18     $P_t \leftarrow$ InputPointCloud($W_{t-1}$, $S_t$, $k_t$)
19     $W_t \leftarrow$ gradient descent update using previous weights $W_{t-1}$ and input $P_t$
20 end
21 return $W_T$
```

curtain data (Table 5.3 rows 2-4) decreases substantially to less 2%, since this data distribution diverges during training. However, the performance on single-beam LiDAR remains relatively same, since the LiDAR data distribution doesn't change. This demonstrates the importance of re-generating the training data online as the weights of the detector change.

	Virtual KITTI			
	3D mAP		BEV mAP	
	0.5 IoU	0.7 IoU	0.5 IoU	0.7 IoU
Single Beam Lidar	**37.68**	**18.65**	**38.14**	**30.08**
1 Light Curtain	1.41	0.48	1.61	0.75
2 Light Curtains	0.73	0.38	1.22	0.58
3 Light Curtains	0.68	0.36	1.13	0.54

Table 5.3: Performance of the detector trained with single-beam LiDAR and up to three light curtains, without online training data generation. The training dataset is collected using the initial weights of the network and is fixed during the remainder of training. The light curtain performance decreases substantially.

5.6 Noise simulations

In order to simulate noise in the real-world sensor, we add 10% noise to the light curtain input, for varying number of light curtain placements, on the Virtual KITTI dataset. The results are shown in Table 5.4. The results are comparable to without noise, indicating that our method is robust to noise and is likely to transfer well to real-world data.

	Virtual KITTI							
	Without noise				**With noise**			
	3D mAP		BEV mAP		3D mAP		BEV mAP	
	0.5 IoU	0.7 IoU	0.5 IoU	0.7 IoU	0.5 IoU	0.7 IoU	0.5 IoU	0.7 IoU
Single Beam Lidar	39.91	15.49	40.77	36.54	39.03	17.13	39.93	30.26
1 Light Curtain	58.01	35.29	58.51	47.05	57.04	25.99	57.65	45.31
2 Light Curtains	60.86	37.91	61.10	49.84	59.43	30.91	59.89	46.11
3 Light Curtains	**68.52**	**38.47**	**68.82**	**50.53**	**60.02**	**31.09**	**66.78**	**46.39**

Table 5.4: Performance of detectors trained with single-beam LiDAR and up to three light curtains, with 10% additional noise in the light curtain input. Performance is not significantly lower than without noise.

5.7 Efficiency analysis

In this section, we report the time taken by our method, for varying number of light curtain placements, and for different light curtain placement algorithms, in Table 5.5. The time (in seconds) includes the time taken for all preceding steps. For example, the time for 2 light curtain placements includes the time required for generating the single-beam LiDAR data, computing the optimal first and second light curtain placements, and all intermediate forwarded passes through the detection network while generating uncertainty maps. The time is averaged over 100 independent trials over different scenes, and we report the 95% confidence intervals.

Note that as we place more light curtains, more time is consumed for the network's forward pass and in calculating where to place the light curtain. This presents a speed-accuracy tradeoff; more light curtains will improve detection accuracy at the expense of taking more

	Single-beam LiDAR	One light curtain	Two light curtain	Three light curtain
Random	0.096 ± 0.001	0.763 ± 0.008	1.441 ± 0.014	2.133 ± 0.014
Fixed depth - 15m	0.090 ± 0.002	0.765 ± 0.008	1.412 ± 0.012	2.028 ± 0.018
Fixed depth - 30m	0.095 ± 0.002	0.789 ± 0.005	1.474 ± 0.008	2.180 ± 0.013
Fixed depth - 45m	0.094 ± 0.001	0.778 ± 0.003	1.475 ± 0.013	2.174 ± 0.012
Greedy Optimization (Randomly break ties)	0.092 ± 0.000	0.825 ± 0.014	1.547 ± 0.023	2.250 ± 0.030
Greedy Optimization (Min laser angle change)	0.086 ± 0.001	0.824 ± 0.010	1.543 ± 0.020	2.242 ± 0.028
Frontoparallel + Uncertainty	0.091 ± 0.001	0.441 ± 0.003	0.807 ± 0.006	1.165 ± 0.008
Dynamic Programming	0.097 ± 0.008	0.944 ± 0.010	1.767 ± 0.015	2.600 ± 0.020

Table 5.5: Time efficiency (in seconds) for varying number of light curtains and different light curtain placement algorithms. Time is averaged over 100 independent trials over different scenes, and we report the 95% confidence intervals.

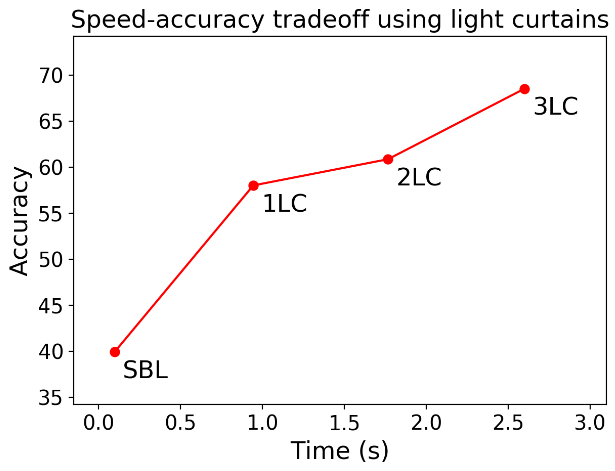


Figure 5.6: Speed-accuracy tradeoff using light curtains optimized by dynamic programming, on the Virtual KITTI dataset. More light curtains correpsond to increased accuracy but reduced speed.

time. On the other hand, our method can run faster using fewer light curtains but with a decreased accuracy. This tradeoff is visualized in Figure 5.6.

5.8 Conclusions

In this work, we develop a method to use light curtains, an actively controllable resource-efficient sensor, for object recognition in static scenes. We propose to use a 3D object detector's prediction uncertainty as a guide for deciding where to sense. By encoding the constraints of the light curtain into a graph, we show how to optimally and feasibly place a light curtain that maximizes the coverage of uncertain regions. We are able to train an active detector that interacts with light curtains to iteratively and efficiently sense parts of scene in an uncertainty-guided manner, successively improving detection accuracy. We hope this work pushes towards replacing expensive multi-beam LiDAR systems with inexpensive controllable sensors, enabled by designing perception algorithms for autonomous driving that integrate sensing and recognition.

CHAPTER 6

Active Depth Estimation

6.1 Introduction

Spinning fixed scan LiDARs have been the de-facto sensor of choice in safety critical systems such as autonomous driving, due to their reliability in depth estimation. However, their reduced spatial-resolution, slow speed and their prohibitive cost has made en-masse adoption in personal vehicles hard. To counter these issues, depth estimation from RGB cameras has been heavily researched. However, issues such as oversaturation, feature correspondence errors and scale ambiguity has made relying on these sensors unsafe.

To capture the error and uncertainty in RGB-only depth estimation, previous work had formulated that task as a probabilistic regression problem, by predicting per-pixel depth distributions via a "Depth Probability Volume (DPV)" [LGK+19] [CC18] [YHR19]. The DPV provides both a maximum likelihood-estimate (MLE) of the depth map, as well as the corresponding per-pixel uncertainty measure. However, these works do not adaptively or physically correct for this uncertainty, instead relying purely on multi-view camera constraints for passive correction.

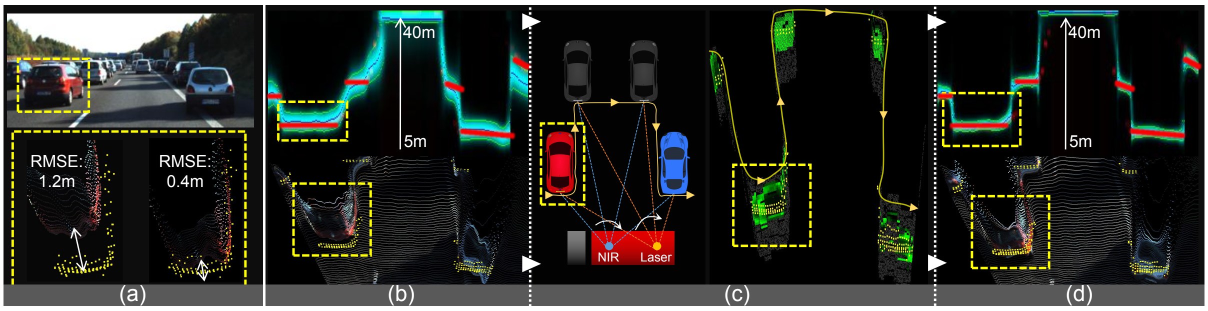


Figure 6.1: **(a)** We show how errors in Monocular Depth Estimation are corrected when used in tandem with an Adaptive Sensor such as a Triangulating Light Curtain (Yellow Points and Red lines are Ground Truth). **(b)** We predict a per-pixel Depth Probability Volume from Monocular RGB and we observe large per-pixel uncertainties ($\sigma = 3m$) as seen in the Bird's Eye View / Top-Down Uncertainty Field slice. **(c)** We actively drive the Light Curtain sensor's Laser to exploit and sense multiple regions along a curve that maximize information gained. **(d)** We feed these measurements back recursively to get a refined depth estimate, along with a reduction in uncertainty ($\sigma = 1m$).

In this chapter, we devise a framework that *adaptively* exploits the depth uncertainty in a per-pixel DPV from RGB images and refines it using programmable light curtains [BWW+19, WBW+18].

We begin by formulating an iterative Bayesian inference approach to adaptive depth sensing using only the light curtain (LC). This is done by building and adapting the 3D DPV representation as a collapsible 2D Uncertainty Field (UF), formulating a probabilistic depth representation of the sensor model and building planning and sensing policies within the sensor constraints. We then build a deep learning architecture that can generate a similar DPV from Monocular or Stereo RGB inputs, and use that as a prior for adaptive sensing. We then fuse the LC measurements back into our network to get a refined depth estimate (see Fig. 6.1).

We conducted experiments of adaptive depth sensing from the LC alone by starting with a Gaussian prior, and showed convergence to true depth with enough iterations. We then trained a network to predict depth distributions from RGB images, used that as a prior for sensing, and fed those new LC measurements back to the network. Through extensive experiments with a simulated LC (with KITTI dataset [GLSU13]) and sensors in the real-world, we show significant speedup in depth convergence and increased accuracy (see Fig. 6.1). As a result, our method has the potential of being a higher resolution, lower cost alternative to a LiDAR.

6.2 Sensor setup

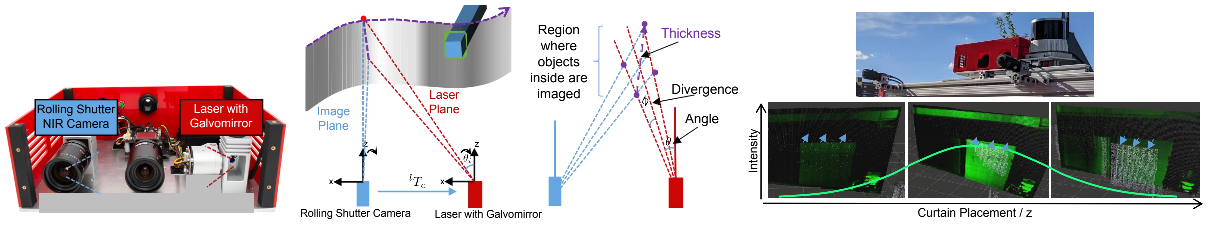


Figure 6.2: **Left:** Our adaptive sensor of choice, the Triangulation Light Curtain (LC) [BWW+19], consists of a laser line, Galvomirror and NIR camera. **Middle:** The light curtain senses a ruled 3D surface extruding from a given top-down 2D curve we call a curtain. Surfaces within the *thickness* of the curtain, result in higher intensity in the NIR image. **Right:** A planar curtain swept across various depths. As the curtain plane approaches the true surface, the measured intensity increases, due to the sensing location and curtain thickness. Above that we show our real-world sensor setup.

The Light Curtain device (Fig 6.2) consists of a rolling shutter Near-Infrared (NIR) camera rotated 90° (that images planes in the world per pixel column), a Line Laser module and a Galvomirror (that generates planes of light depending on the angle). The exact sensing location is obtained by intersecting (triangulating) the imaging and laser planes. Sweeping this laser line creates a 3D ruled surface called a curtain. We can place a curtain along any surface by controlling the galvo and rolling shutter speed subject to it's physical constraints, making the sensor adaptive in nature. Note that the image and laser planes have some divergence,

The project webpage for this chapter can be found at: https://soulslicer.github.io/rgb-lc-fusion/.
We have open sourced our code for active depth estimation using programmable light curtains at: https://github.com/CMU-Light-Curtains/DepthEstimation.

so their intersection results in a volume in space (bounded by purple points in Fig. 6.2) with some *thickness*, where any objects that intersect it result in higher intensities in the NIR image. This means that as the sensing location approaches the true surface, pixel intensities on NIR image increases.

Real-world experiments are conducted using our array of sensors consisting of an RGB Stereo Camera Pair, the Light Curtain device, and a 128-beam Lidar for accuracy validation and RGB depth estimation network training. Simulated experiments are also conducted with KITTI dataset [GLU12], through a Light Curtain Simulator that uses the ground truth depth map along with the ability to vary NIR instrinsics, laser extrinsincs, Galvomirror speed and laser divergence/thickness and angle.

6.3 Depth from light curtain only

Before considering RGB + Light Curtain fusion (sec. 6.5), we begin by focusing on the problem of adaptively discovering the depth of a scene using only the light curtain.

6.3.1 Representation

We wish to estimate the depth map $\mathbf{D} = \{d_{u,v}\}$ of the scene, which specifies the depth value $d_{u,v}$ for every camera pixel (u, v) at spatial resolution [H,W]. Since there is inherent uncertainty in the depth value at every pixel, we represent a *probability distribution* over depths for every pixel. Let us define $\mathbf{d}_{u,v}$ to be a *random variable* for depth predictions at the pixel (u, v). We quantize depth values into a set $\mathcal{D} = \{d_0, \ldots, d_{N-1}\}$ of N discrete, uniformly spaced depth values lying in $(d_{\text{min}}, d_{\text{max}})$. All the predictions $\mathbf{d}_{u,v} \in \mathcal{D}$ belong to this set. The output of our depth estimation method for each pixel is a probability distribution $P(\mathbf{d}_{u,v})$, modeled as a categorical distribution over $\mathcal{D}$. In this work, we use $N = 64$, resulting in a Depth Probability Volume (DPV) tensor of size [64, W, H]:

$$\mathcal{D} = \{d_0, \ldots, d_{N-1}\}; d_q = d_{\text{min}} + (d_{\text{max}} - d_{\text{min}}) \cdot q \tag{6.1}$$

$$\sum_{q=0}^{N-1} P(\mathbf{d}_{u,v} = d_q) = 1 \text{ (q is the quantization index)} \tag{6.2}$$

$$\text{Depth estimate } = \mathbb{E}[\mathbf{d}_{u,v}] = \sum_{q=0}^{N-1} P(\mathbf{d}_{u,v} = d_q) \cdot d_q \tag{6.3}$$

This DPV can be initialized using another sensor such as an RGB camera, or can be initialized with a Uniform or Gaussian distribution with a large σ for each pixel.

While an ideal sensor could choose to plan a path to sample the full 3D volume, our light curtain device only has control over a top-down 2D profile. Hence, we compress our DPV into a top-down an "Uncertainty Field" (UF) [YHR19], by averaging the probabilities of the DPV across a subset of each column (Fig. 6.3). This subset considers those pixels (u, v) whose corresponding 3D heights $h(u, v)$ are between $(h_{\text{min}}, h_{\text{max}})$. The UF is defined for the camera column u and quantized depth location q as:

$$UF(u, q) = \frac{1}{|\mathcal{V}(u)|} \sum_{v \in \mathcal{V}(u)} P(\mathbf{d}_{u,v} = d_q)$$
$$\text{where } \mathcal{V}(u) = \{v \mid h_{\text{min}} \leq h(u, v) \leq h_{\text{max}}\} \tag{6.4}$$

We denote the categorical distribution of the uncertainty field on the u-th camera ray as: $UF(u) = \text{Categorical}(d_q \in \mathcal{D} \mid P(d_q) = UF(u,q))$.

6.3.2 Light Curtain Optimization

We can use the extracted Uncertainty Field (UF) to compute the best light curtain to place. We use the constraint graph framework introduced in Section 4.1. A single light curtain placement is defined by a set of control points $\{q(u)\}_{u=1}^W$, where u indexes columns of the camera image of width W, and $0 \le q(u) \le N-1$. This denotes that the curtain intersects the camera rays of the u-th column at the discretized depth $d_{q(u)} \in \mathcal{D}$. We wish to maximize the objective $J(\{q(u)\}_{u=1}^W) = \sum_{u=1}^W UF(u, q(u))$. Let $\mathbf{X}_u$ be the 2D point in the top-down view that corresponds to the depth $q(u)$ on camera rays of column u. The control points $\{q(u)\}_{u=1}^W$ must be chosen to satisfy the physical constraints of the light curtain device (see equations 3.1, 3.2).

$$\begin{aligned}
&\arg\max_{\{q(u)\}_{u=1}^W} \sum_{u=1}^W UF(u, q(u)) \\
&\text{such that } |\theta(\mathbf{X}_u) - \theta(\mathbf{X}_{u-1})| \le \omega_{\max}\, \Delta t,\ \forall 2 \le u \le W \\
&\qquad\qquad |\theta(\mathbf{X}_{u+1}) + \theta(\mathbf{X}_{u-1}) - 2\theta(\mathbf{X}_u)| \le \alpha_{\max}\, (\Delta t)^2,\ \forall 2 \le u < W
\end{aligned} \tag{6.5}$$

We use the dyanmic programming approach outlined in Section 4.2 to compute the curtain that maximizes the objective while satisfying the constraints.

6.3.3 Curtain Placement

The uncertainty field UF contains the current uncertainty about pixel-wise object depths $\mathbf{d}_{u,v}$ in the scene. Let us denote by $\pi(d^{c_k} \mid UF)$ the placement policy of the k-th light curtain, where $d^{c_k} = \{d^{c_k}_{u,v} \mid \forall u, v\}$. Our goal is to sample light curtain placements $d^{c_k} \sim \pi(d^{c_k} \mid UF)$ from this policy, and obtain intensities $i_{u,v}$ for every pixel.

To do this, we propose two policies: π_0 and π_1. In Fig. 6.4, we have placed a single curtain along the highest probability region per column of rays, but our goal is to maximize the information gained. For this, we generate corresponding entropy fields $H(u,q)_i$ to be input to the planner computed from $UF(u,q)$. We use two approaches to generate $H(u,q)$: π_0 finds

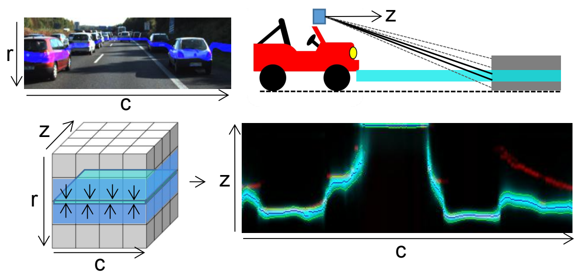


Figure 6.3: Our state space consists of a Depth Probability Volume (DPV) (left) storing per-pixel uncertainty distributions. It can be collapsed to a Bird's Eye Uncertainty Field (UF) (right) by averaging those rays in each row (blue pixels) of the DPV that correspond to a slice on the road parallel to the ground plane (right) (cyan pixels). Red pixels on UF represent the low resolution LiDAR ground truth.

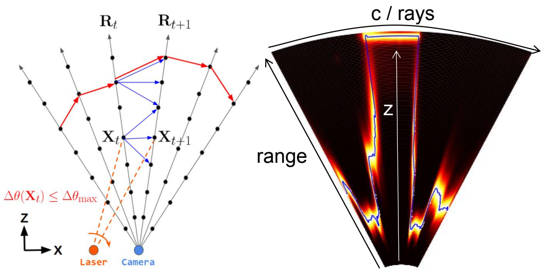


Figure 6.4: Given an Uncertainty Field (UF), our planner solves for an optimal galvomirror trajectory subject to it's constraints (eg. $\mathring{\theta}_{max}$). We show a 3D ruled surface / curtain placed on the highest probability region of UF.

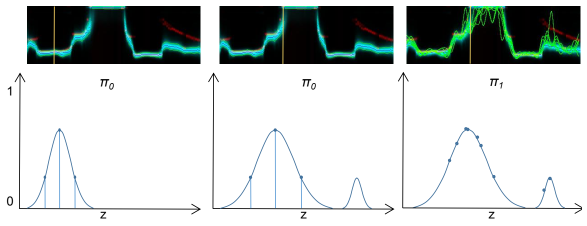


Figure 6.5: Sampling the world at the highest probability region is not enough. To converge to the true depth, we show policies that place additional curtains given UF. Let's look at a ray (in yellow) from the UF to see how each policy works. **Left:** π_0 given a unimodal gaussian with small σ. **Middle:** π_0 given a multimodal gaussian with larger σ. **Right:** π_1 given a multimodal gaussian with larger σ. Observe that π_1 results in curtains being placed on the second mode.

the mean in each ray's distribution $UF(u)$ and selects a σ_{π_0} that determines the neighbouring span selected. π_1 samples a point on the ray given $UF(u)$.

As seen in Fig. 6.5, strategy π_0 is able to generate fields that adaptively place additional curtains around a consistent span around the mean with some σ_{π_0}, but is unable to do so in cases of multimodal distributions. π_1 on the other hand is able to place a curtain around the second modality, albeit with a lower probability. We will show the effects of both strategies in our experiments.

6.3.4 Observation Model

A curtain placement corresponds to specifying the depth for each camera ray indexed by u from the top-down view. After placing the light curtain, intensities $i_{u,v}$ are imaged by the light curtain's camera at every pixel (u, v). The measured intensity at each pixel is a function of the curtain placement depth $d^c_{u,v}$ on that camera ray, the unknown ground truth depth $\mathbf{d}_{u,v}$ of that pixel, the thickness of the light curtain $\sigma(u, v, d^c_{u,v})$ for a particular pixel and curtain placement, and the maximum intensity possible if a curtain is placed perfectly on the surface $p_{u,v}$ (varies from 0 to 1). From real world data Fig. 6.6, we find the intensity decays exponentially as the distance between the curtain placement $d^c_{u.v}$ and ground truth depth $\mathbf{d}_{u,v}$ increases, with the scaling factor $p_{u,v}$ parameterizing the surface properties. We also simulate sensor noise as a Gaussian distribution with standard deviation σ_{nse}. The overall sensor model

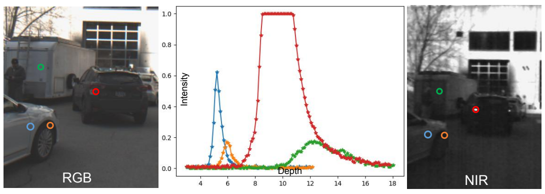


Figure 6.6: We sweep a planar light curtain across a scene at 0.1m intervals, and observe that the changes in intensity over various pixels follow an exponential falloff model. **Blue / Orange**: Car door has a higher response than the tire. **Green**: Object further away has a lower response with a larger sigma due to curtain thickness. **Red**: Retroreflective objects cause the signal to saturate.

$P(i_{u,v} \mid \mathbf{d}_{u,v}, d^c_{u,v})$ can be described as:

$$P(i_{u,v} \mid \mathbf{d}_{u,v}, d^c_{u,v}) \equiv \mathcal{N}\left(i_{u,v} \mid \exp\left(-\left(\frac{d^c_{u,v} - \mathbf{d}_{u,v}}{\sigma(u,v,d^c_{u,v})}\right)^2\right) . p_{u,v}, \sigma^2_{\textsf{nse}}\right) \tag{6.6}$$

Note that when $d^c_{u,v} = \mathbf{d}_{u,v}$ and $p_{u,v} = 1$, the mean intensity is 1 (the value), and it reduces exponentially as the light curtain is placed farther from the true surface. $p_{u,v}$ can be extracted from the ambient NIR image.

6.3.5 Recursive Bayesian Update

How do we incorporate the newly acquired information about the scene from the light curtain to update our current beliefs of object depths? Since we have a probabilistic sensor model, we use the Bayes' rule to infer the posterior distribution of the ground truth depths given the observations. Let $P_{\textsf{prev}}(u, v, q)$ denote the probability of the depth at pixel (u, v) being equal to d_q before sensing, and $P_{\textsf{next}}(u, v, q)$ the updated probability after sensing. Then by Bayes' rule:

$$\begin{aligned} &P_{\textsf{next}}(u,v,q) \\ &= P(\mathbf{d}_{u,v} = d_q \mid i_{u,v}, d^{c_k}_{u,v}) \\ &= \frac{P(\mathbf{d}_{u,v} = d_q) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})}{P(i_{u,v} \mid d^{c_k}_{u,v})} \\ &= \frac{P(\mathbf{d}_{u,v} = d_q) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})}{\sum_{q'=0}^{N-1} P(\mathbf{d}_{u,v} = d_{q'}) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_{q'}, d^{c_k}_{u,v})} \\ &= \frac{P_{\textsf{prev}}(u,v,q) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})}{\sum_{q'=0}^{N-1} P_{\textsf{prev}}(u,v,q') \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_{q'}, d^{c_k}_{u,v})} \end{aligned} \tag{6.7}$$

Note that $P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})$ is the sensor model whose form is given in Equation 6.6.

If we place K light curtains at a given time-step, we can incorporate the information received from all of them into our Bayesian update simultaneously. Since the sensor noise is independent of curtain placement, the likelihoods of the observed intensities can be multiplied across the

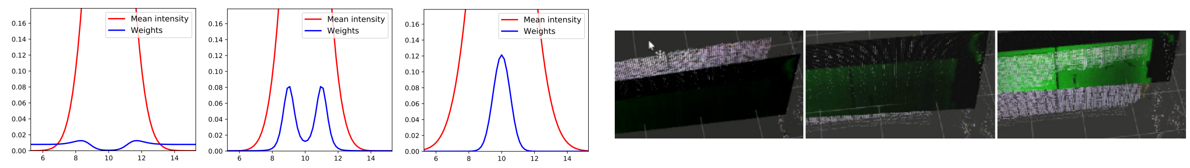


Figure 6.7: Visualization of the recursive Bayesian update method to refine depth probabilities after observing light curtain intensities. The curtain is placed at 10m. The red curves denote the expected intensity (Y-axis) as a function of ground truth depth (X-axis); this is the sensor model given in Eqn. 6.6. After an intensity is observed by the light curtain, we can update the probability distribution of what the ground truth depth might be using our sensor model and the Bayes' rule. The updated probability is shown by the blue curves, computed using the Bayesian update of Eqn. 6.7 (here, the prior distribution P_{prev} is assumed to be uniform, and $d^{c_k}_{u,v} = 10m$). **Left:** Low i return leads to an inverted Gaussian distribution at the light curtain's placement location, with other regions getting a uniform probability. **Middle:** Medium i means that the curtain isn't placed exactly on the object and the true depth could be on either side of the light curtain. **Right:** High i leads to an increased belief that the true depth is at 10m.

curtains. Hence, the overall update becomes:

$$P_{\text{next}}(u,v,q) \\ = \frac{P_{\text{prev}}(u,v,q)\cdot\prod_{k=1}^{K} P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})}{\sum_{q'=0}^{N-1} P_{\text{prev}}(u,v,q')\cdot\prod_{k=1}^{K} P(i_{u,v} \mid \mathbf{d}_{u,v} = d_{q'}, d^{c_k}_{u,v})}$$

The behavior of this model as the placement depth $d^c_{u,v}$, curtain thickness $\sigma(u,v,d^c_{u,v})$ and intensity i change is seen in Fig. 6.7. We observe that low intensities lead to an *inverting gaussian* like weight updates, with a low weight at the light curtain's placement location while other regions get uniform weights. This indicates that the method is certain that an object doesn't exist at the light curtain's location, but is uniformly uncertain about the other un-measured regions. A medium intensity leads due a bimodal gaussian, indicating that the curtain may not be placed exactly on the surface and could be on either side of the curtain. Finally, as the intensity rises, so does weight assigned to the light curtain's placement location.

6.4 Experiments with light curtain only

We first demonstrate depth estimation using just the Light Curtain as described in Sec. 6.3. In this initial baseline, we track the Uncertainty Field (UF) depth error by computing the RMSE error metric $\sqrt{[i = 1]n\sum\frac{(\mathbb{E}(UF(u,q))-\mathbf{d_{gt}}(u)_i)^2}{n}}$ against ground truth. We evaluate our method against several outdoor scenarios consisting of vehicles in a scene.

Planar Sweep Curtain Placement: We are able to simulate the light curtain response using depth from LiDAR. A simple fixed policy not adapted to the UF helps validate our sensor model and provides corroboration between the simulated and real light curtains. We perform a uniform sweep across the scene above (at 0.25 to 1.0m intervals) (Fig. 6.8), incorporating intensity measurements at each pixel for each curtain using our process described earlier. Our simulated device is able to reasonably match the real device, and we also show how sweeping more curtains increases accuracy at the cost of increased runtime (Table. 6.1).

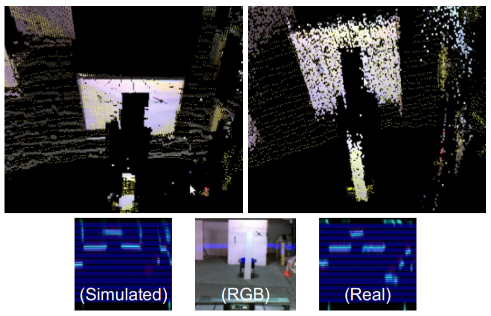


Figure 6.8: We demonstrate corroboration between simulated and real light curtain device by sweeping several planes across this scene. Colored point cloud is the estimated depth, and lidar ground truth in yellow. **Left:** LC simulated from the lidar depth. **Right:** Using the real device.

Policy	50LC @ 0.25m	25LC @ 0.5m	50LC @ 0.25m	25LC @ 0.5m	12LC @ 1.0m
RMS/m	1.156	1.374	1.284	1.574	1.927
Runtime /s	-	-	2	1	0.5

Table 6.1: Policy depicts different numbers of light curtains (LC) placed at regular intervals. The first two columns are simulations and the rest are real experiments. Sampling the scene by placing more curtains results in better depth accuracy (lower RMS) at the cost of higher runtime.

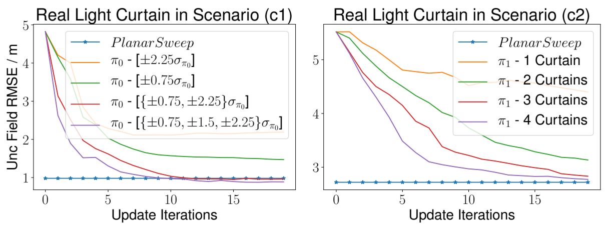


Figure 6.9: Curtain placement as a function of the Uncertainty Field (UF) converges within a lower number of iterations as opposed to a uniform planar sweep which took 25 iterations

Policy based Curtain Placement: Sweeping a planar LC can be time consuming (25 iterations), so we want our curtains to be a function of our UF. We evaluated two different scenarios (c1, c2) for each placement policy (π_0, π_1), and we observed that planning and placing curtains as a function of UF results in much faster convergence (Fig. 6.9).

6.5 Depth from light curtain + RGB fusion

While starting from a uniform or Gaussian prior with a large uncertainty is a valid option, it is slow to converge. Furthermore, a light curtain's only means of depth estimation is extracted primarily along the ruled placement of the curtain, at least based on our above placement policies. We would ideally like to use information from a Monocular RGB camera or Stereo Pair to initialize our prior, with a similar DPV representation. For this, a Deep Learning based architecture is ideal, and we also reason that such an architecture could potentially learn to fuse/incorporate information from both modalities better.

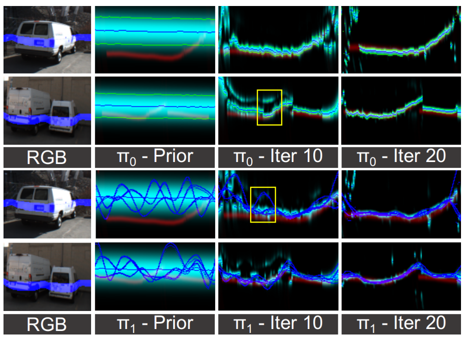


Figure 6.10: Looking at the top-down Uncertainty Field (UF), we see per pixel distributions in Cyan and the GT in Red. We start with a gaussian prior with a large σ, take measurements and apply the bayesian update, trying both policies π_0 and π_1. Note how measurements taken close to the true surface split into a bimodal distribution (Yellow Box)

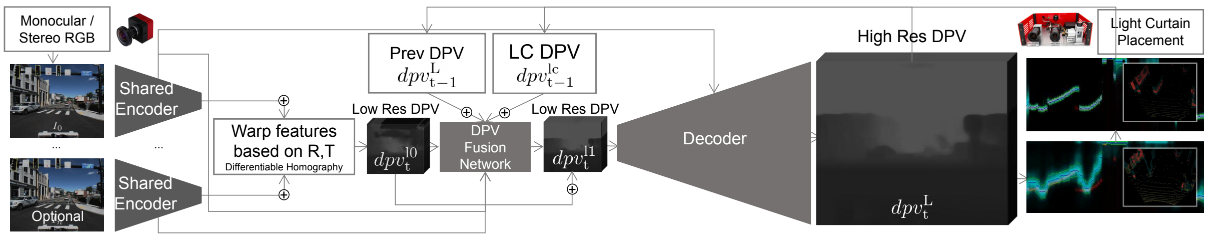


Figure 6.11: Our Light Curtain (LC) Fusion Network can take in RGB images from a single monocular image, multiple temporally consistent monocular images, or a stereo camera pair to generate a Depth Probability Volume (DPV) prior. We then recursively drive our Triangulation Light Curtain's laser line to plan and place curtains on regions that are uncertain and refine them. This is then fed back on the next timestep to get much more refined DPV estimate.

6.5.1 Structure of Network

The first step is to build a network (Fig. 6.11) that can generate DPV's from RGB images. We extend the Neural-RGBD [LGK+19] architecture to incorporate light curtain measurements. Anywhere from 1 to N images, usually two (I_0, I_1), are fed into shared encoders, and the features are then warped into different fronto-parallel planes of the reference image I_0 using pre-computed camera extrinsics $R^{I_1}_{I_o}, t^{I_1}_{I_o}$. Further convolutions are run to generate a low resolution DPV dpv^{l0}_{t} [H/4, W/4] where the log softmax operator is applied and regressed on. The transformation between the cameras acts as a constraint, forcing the feature maps to respect depth to channel correspondence. The add operator into a common feature map is similar to adding probabilities in log-space.

dpv^{l0}_{t} is then fed into the DPV Fusion Network (a set of 3D Convolutions) that incorporate a downsampled version of dpv^{L}_{t-1} along with the the light curtain DPV that we had applied recursive Bayesian updates on dpv^{lc}_{t-1}, and a residual is computed and added back to dpv^{l0}_{t} to generate dpv^{l1}_{t} to be regressed upon similarly. With a 30% probability, we train without dpv^{lc}_{t-1} feedback by inputting a uniform distribution. Finally, dpv^{l1}_{t} is then passed into a decoder with skip connections to generate a high resolution DPV dpv^{L}_{t}. This is then used to plan and place light curtains, from which we generate a new dpv^{lc}_{t} to be fed into the next stage.

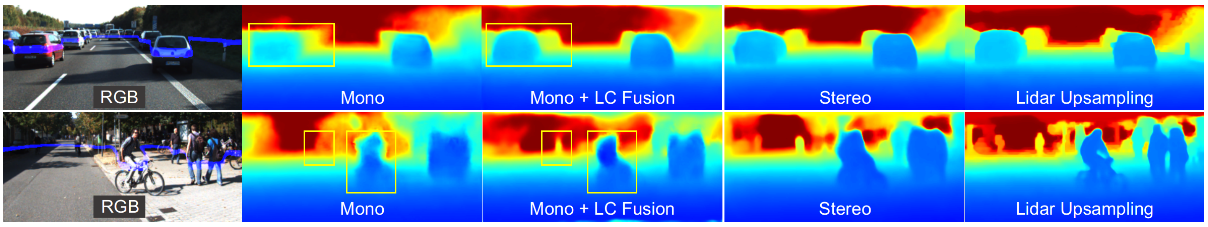


Figure 6.12: In KITTI + Simulated Light Curtain, we note improved depthmaps when Monocular inputs are fused with Light Curtain inputs. Note the improvements in regions bounded in the yellow box. Our network is also capable of ingesting Stereo inputs, and also solving the task of Lidar Upsampling

6.5.2 Loss Functions

Soft Cross Entropy Loss: We build upon the ideas in [YHR19] and use a soft cross entropy loss function, with the ground truth LiDAR depthmap becoming a Gaussian DPV with σ_{gt} instead of a one hot vector. This way, when estimating $\mathbb{E}\left(dpv^{gt}\right)$ we get the exact depth value instead of an approximations limited by the depth quantization $\mathcal{D}$. We also make the quantization slightly non-linear to have more steps between objects that are closer to the camera:

$$l_{sce} = \frac{-\sum_i \sum_d \left(dpv^{\{\mathrm{l0,l1,L}\}} * log\left(dpv^{gt}\right)\right)}{n} \tag{6.8}$$

$$\mathcal{D} = \{d_0, \ldots, d_{N-1}\}; d_q = d_{\mathsf{min}} + (d_{\mathsf{max}} - d_{\mathsf{min}}) \cdot q^{pow} \tag{6.9}$$

L/R Consistency Loss: We train on both the Left and Right Images of the stereo pair whose Projection matrices P_l, P_r are known [GAB17]. We enforce predicted Depth and RGB consistency by warping the Left Depthmap into the Right Camera and vice-versa, and minimize the following metric:

$$D_l = \mathbb{E}\left(dpv_l^L\right) \qquad D_r = \mathbb{E}\left(dpv_r^L\right) \tag{6.10}$$

$$l_{dcl} = \frac{1}{n} \sum_i \left(\frac{\left| D_{\{l,r\}} - w\left(D_{\{r,l\}}, P_{\{l,r\}}\right)\right|}{D_{\{l,r\}} + w\left(D_{\{r,l\}}, P_{\{l,r\}}\right)} \right) \tag{6.11}$$

$$l_{rcl} = \frac{1}{n} \sum_i \left(||I_{\{l,r\}} - w\left(I_{\{r,l\}}, D_{\{l,r\}}, P_{\{l,r\}}\right) ||_1 \right) \tag{6.12}$$

Edge aware Smoothness Loss: We ensure that neighbouring pixels have consistent surface normals, except on the edges/boundaries of objects with the Sobel operator S_x, S_y via the term:

$$l_s = \frac{1}{n} \sum_i \left(\left| \frac{\partial I}{\partial x} \right| e^{-|S_x I|} + \left| \frac{\partial I}{\partial y} \right| e^{-|S_y I|} \right) \tag{6.13}$$

6.6 Experiments with light curtain + RGB fusion

We train and validate our algorithms on the KITTI dataset. We then trained the same network by initializing on those weights, but using our custom dataset to evaluate our algorithms with the real sensors on the Jeep.

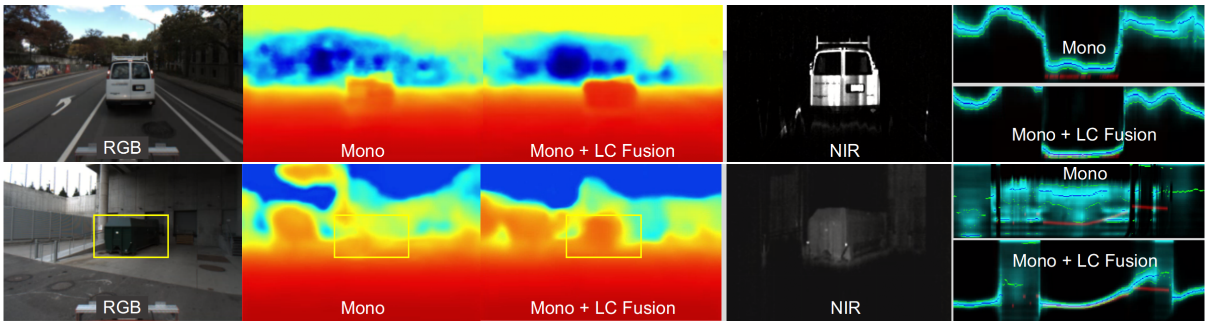


Figure 6.13: In real world Experiments, we are able to see the monocular scale ambiguity in domain specific scenarios (driving scenario with a van 8m away) get corrected by the Light Curtain, and we are able to see correction in an arbitrary scene (dumpster 15m away) provided to the system as well

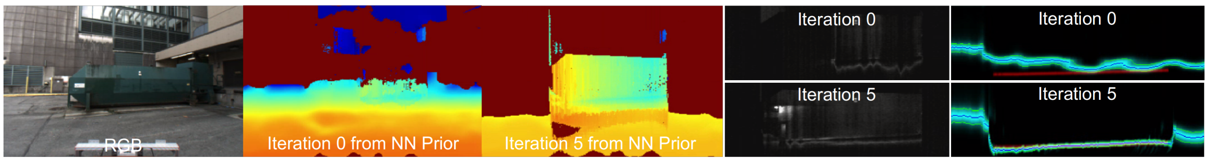


Figure 6.14: We show the internal state of the bayesian update at Iteration 0 and Iteration 5. Starting with a prior DPV from Monocular Depth estimation, we show the convergence of the sensor's laser and curtain profile on an object 10m away

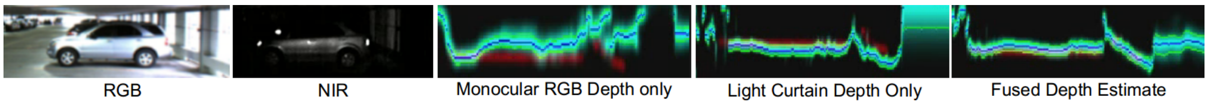


Figure 6.15: Monocular RGB alone suffers from scale ambiguity but does give an inital uncertain depth estimate on a car 15m away. Iterating on Light Curtain measurements from a mean-centered gaussian prior alone gives a more accurate depth but with a noisy profile, but starting with the RGB DPV results in a more accurate and smoother profile.

For evaluation, we consider the RMSE metric against the entire depthmap as opposed to just the Uncertainty Field (UF) as $\sqrt{[i = 1]n\sum\frac{(\mathbb{E}(\mathbf{d}_{u,v}) - \mathbf{d_{gt}}(u,v)_i)^2}{n}}$ against our ground truth.

DPV Prior from RGB: Our first goal is to ensure that our network is capable of generating a reasonable DPV with monocular RGB input, given the above loss functions. We do some simple experiments that explore these effects.

Parameters	$\sigma_{gt} = 0.05$	$\sigma_{gt} = 0.2$	$\sigma_{gt} = 0.3$	$\sigma_{gt} = 0.3$ with l_{dcl}, l_{rcl}	$\sigma_{gt} = 0.3$ with l_{dcl}, l_{rcl}, l_s
RMSE/m	3.24	3.16	3.06	2.93	2.90

Table 6.2: Effects of Soft Cross Entropy (σ_{gt}), Left/Right Consistency (l_{dcl}, l_{rcl}), Smoothness losses (l_s) on Monocular Depth Estimation.

Table 6.2 shows successively improving performance as we increase σ_{gt}, with poorer performance when the depth is effectively encoded as a one-hot vector (eg. $\sigma_{gt} = 0.05$), since the depth was more likely to be forced into one of the categories in $\mathcal{D}$. Adding in l_{dcl}, l_{rcl} and l_s improved performance further.

Mono vs Stereo

Mono	2.904
Stereo	1.737

Lidar Upsample with DPV Fusion Network

Without DPV Fusion Network	1.118
With DPV Fusion Network	0.702

Table 6.3: **Left:** Stereo pair at t instead of Monocular pair at $t, t-1$ input to the network. **Right:** Fusing the GT LiDAR data with dpv_t^{l0} to generate dpv_t^{l1} and dpv_t^{L} with Bayesian inference vs DPV Fusion Network.

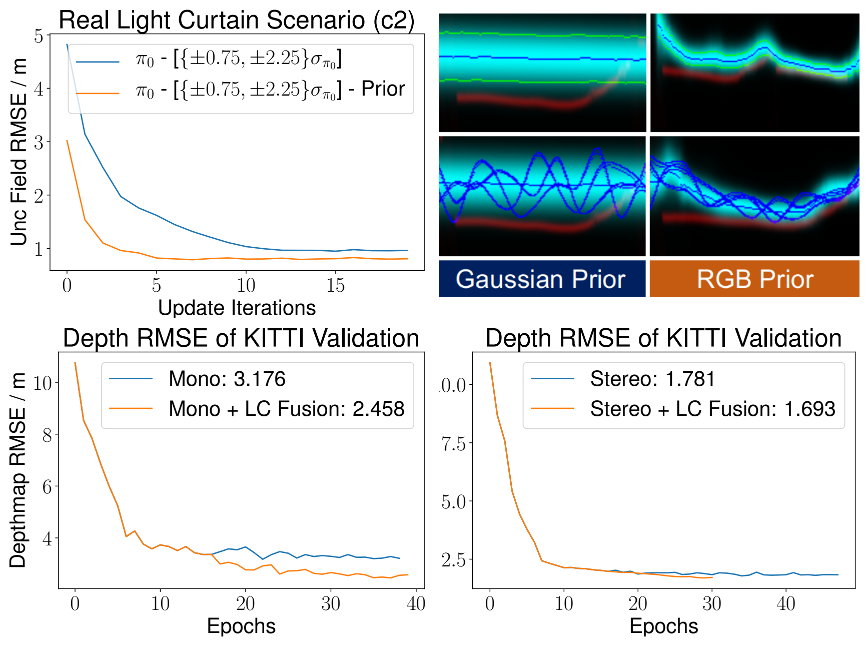


Figure 6.16: Top: Adaptive depth sensing with the Light Curtain: Starting from a Prior distribution from a Monocular Depth Network as opposed to a gaussian with a large σ leads to faster convergence towards the true depth. Bottom: Monocular (Left) and Stereo (Right) Depth Estimation show improvements when we enabled feedback of the sensed Light Curtain DPV at epoch 16 when training on KITTI dataset with light curtain simulator.

Stereo Inputs: Since our method can generalize to any N camera setup, we compare and contrasted monocular pair inputs at times $t, t-1$, against a stereo pair at time t as input (extrinsics known in both cases). As expected, we note significantly better performance with stereo input (Table 6.3).

Effect of a Stronger Prior: Previously, we had run our adaptive sensing algorithm from a gaussian prior with a large σ (Fig. 6.10). In various outdoor experiments, we show that a prior DPV from our network instead, yields higher accuracy and faster convergence towards the true depth (Fig. 6.15, Fig. 6.16)(a, b).

DPV Fusion Network: With this corrected DPV, we want to explore how to effectively handle erroneous measurements (due to low light curtain returns etc.), or fuse it other DPVs (from previous frame or from another sensor). With this in mind, we consider the sub task of LiDAR Upsampling. The Velodyne LiDAR in the KITTI dataset, can be converted into a low resolution depthmap, and consequently a low-res DPV we call dpv_t^{gt}. We could then fuse both dpv_t^{l0} and dpv_t^{gt} to generate dpv_t^{l1} using Bayesian inference. Alternatively, we could feed both of those inputs into our DPV Fusion Network, which relies on a series of 3D Convolutions. We note improved performance in this upsampling task using this approach as seen in Table 6.3.

Light Curtain Fusion Network: Finally, we combine all of these concepts into one. Here, we train our monocular and stereo depth estimation without light curtain feedback, and one where we enable dpv_t^{lc} to be planned and fed-back on the next stage via our DPV Fusion Network, as described in (Fig. 6.11). Training is done on the KITTI dataset with our light curtain simulator, with a maximum of 5 update iterations for performance and memory reasons. We observed qualitative (Fig. 6.13) and quantitative (Fig. 6.16(c, d)) performance improvement of depth with Monocular input, and marginal but visible improvement with Stereo. This is due to the wide baseline of 0.7m in the stereo pair, so we could see that smaller baseline pairs would benefit more with our light curtain measurements.

Performance: Our un-optimized implementation of each planning and curtain placement step takes 40ms. Depth convergence occurs in 5 iterations (5 fps) when starting from a monocular RGB prior and 10 iterations (2.5 fps) with a Gaussian prior. In temporally continuous operations, the prior from $t-1$ reduces convergence to 2 iterations (12.5 fps) depending on the camera motion. A well-engineered implementation could achieve 20-40 fps but much faster motion would require explicitly encoding 3D optical flow.

CHAPTER 7

Active Safety Envelopes

7.1 Introduction

Consider a robot navigating in an unknown environment. The environment may contain objects that are arbitrarily distributed, whose motion is haphazard, and that may enter and leave the environment in an undetermined manner. This situation is commonly encountered in a variety of robotics tasks such as autonomous driving, indoor and outdoor robot navigation, mobile robotics, and robot delivery. How do we ensure that the robot moves safely in this environment and avoids collision with obstacles whose locations are unknown a priori? What guarantees can we provide about its perception system being able to discover these obstacles?

In this work, we propose to use light curtains to estimate the “safety envelope" of a scene. We define the safety envelope as an imaginary, vertically ruled surface that separates the robot from all obstacles in the scene. The region between the envelope and the robot is free space and is safe for the robot to occupy without colliding with any objects. Furthermore, the safety envelope “hugs" the closest object surfaces to maximize the amount of free space between the robot and the envelope. More formally, we define a safety envelope as a 1D depth map that is computed from a full 2D depth map by selecting the closest depth value along each column of the 2D depth map (ignoring points on the ground or above a maximal height). As long as the robot never intersects the safety envelope, it is guaranteed to not collide with any obstacle.

Realizing this concept requires addressing two novel and challenging questions: First, where do we place the curtains without a priori knowledge of objects in the scene? The light curtain will only sense the parts of the scene where the curtain is placed. Second, how do we evolve these curtains over time to capture dynamic objects? One approach is to place light curtains at random locations in the unknown scene. Previous work [BWW+19] has empirically shown that random light curtains can quickly discover unknown objects. In this work, we develop a systematic framework to generate random curtains that respect the physical constraints of the light curtain device. Importantly, we develop a method that produces theoretical guarantees on the probability of random curtains (from a given distribution) to detect unknown objects in

The project webpage for this chapter can be found at:
https://siddancha.github.io/projects/active-safety-envelopes-with-guarantees.
We have open sourced our code for active safety envelopes using programmable light curtains at: https://github.com/CMU-Light-Curtains/SafetyEnvelopes.

the environment and discover the safety envelope. Such safety guarantees could be used to certify the efficacy of a robot perception system to detect and avoid obstacles.

Once a part of the safety envelope (such as an object's surface) is discovered, it may be inefficient to keep exploring the scene randomly. Instead, a better strategy is to forecast how the identified safety envelope will move in the next timestep and track it by sensing at the predicted location. We achieve this by training a neural network to forecast the position of the envelope in the next timestep using previous light curtain measurements. However, it is difficult to provide theoretical guarantees for such learning-based systems. We overcome this challenge by combining the deep neural network with random light curtain placements. Using this combination, we are able to estimate the safety envelope efficiently, while furnishing probabilistic guarantees for discovering unknown obstacles. Our contributions are:

1. We develop a dynamic-programming based approach to produce theoretical safety guarantees on the probability of random curtains discovering unknown objects in the environment (Sec. 4.4.2, 7.3.1).
2. We combine random light curtains with a machine learning based forecasting model, trained using imitation learning, to efficiently estimate safety envelopes (Sec. 7.2).
3. We evaluate our approach on (1) a simulated autonomous driving environment, and (2) a real-world environment with moving pedestrians. We empirically demonstrate that our approach consistently outperforms multiple baselines and ablation conditions (Sec. 7.3.2).

7.2 Learning to forecast safety envelopes

Random curtains can help discover the safety envelope of unknown objects in a scene. However, once a part of the envelope is discovered, an efficient way to estimate the safety envelope in future timesteps is to *forecast* how the envelope will move with time and *track* the envelope by placing a new light curtain at the forecasted locations. In this section, we describe how to train a deep neural network to forecast safety envelopes and combine them with random curtains.

Problem setup: We call any algorithm that attempts to forecast the safety envelope as a "forecasting policy". We assume that a forecasting policy is provided with the ground truth safety envelope of the scene in the first timestep. In the real world, this can be done by running one of the less efficient baseline methods once when the light curtain is first started, until the light curtain is initialized. After the initialization, the learning-based method is used for more efficient light curtain tracking. The policy always has access to all previous light curtain measurements. At every timestep, the policy is required to forecast the location of the safety envelope for the next timestep. Then, the next light curtain will be placed at the forecasted location. To leverage the benefits of random curtains that can discover unknown objects, we place random light curtains while the forecasting method predicts the safety envelope of the next timestep. We allow random curtains to override the forecasted curtain: if the random curtain obtains an intensity $\mathbf{I}_t$ on camera ray R_t that is above a threshold τ, the control point of the forecasted curtain for ray R_t is immediately updated to that of the random curtain, i.e. the random curtain overrides the forecasted curtain if the random curtain detects an object. See Appendix 7.5 for details of our efficient, parallelized implementation of random curtain placement and forecasting, as well as an analysis of the pipeline's runtime.

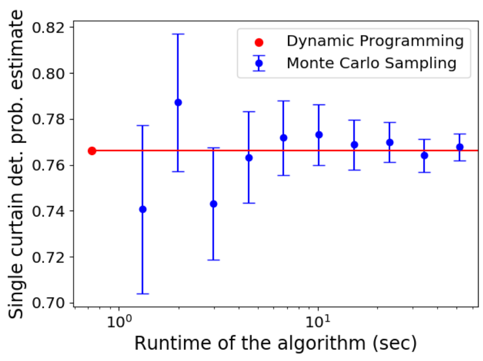
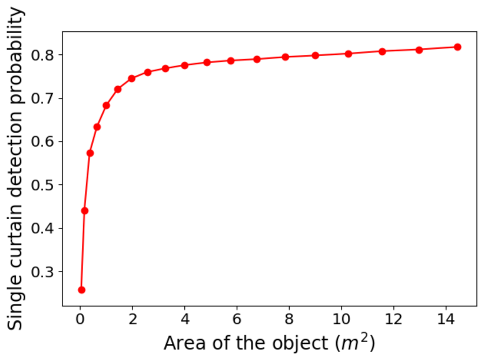


Figure 7.1: Comparison of our dynamic programming approach with a Monte Carlo sampling based approach to estimate the probability of a random curtain detecting an object of size $2m \times 2m$. The x-axis shows the runtime of the methods (in seconds), and the y-axis shows estimated detection probability. Our method (in red) is both *exact* and fast; it quickly produces a single and precise point estimate. Monte Carlo sampling (in blue) samples a large number of random curtain and returns the average number of curtains that intersect the object. The estimate is stochastic and the 95%-confidence intervals are shown in blue. While the Monte Carlo estimate eventually converges to the true probability, it is inherently noisy and orders of magnitude slower than our method.

Handcrafted policy: First, we define a simple, hand-specified light curtain placement policy; this policy will serve both as a baseline and as an input to our neural network, described below. The policy conservatively forecasts a fixed decrease in the depth of the safety envelope for ray R_t if the ray's intensity $\mathbf{I}_t$ is above a threshold (indicative of the presence of an object), and forecasts a fixed increase in depth otherwise. By alternating between increasing and decreasing the depth of the forecasted curtain, this policy can roughly track the safety envelope. However, since the forecasted changes in depth are hand-defined, it is not designed to handle large object motions, nor will it accurately converge to the correct depth for stationary objects.

Neural network forecasting policy: We use a 2D convolutional neural network to forecast safety envelopes in the next timestep. It takes as input (1) the intensities $\mathbf{I}_t$ returned by previous k light curtain placements, (2) the positions of the previous k light curtain placements, and (3) the outputs of the handcrafted policy described above (this helps avoid local minima during training and provides useful information to the network). For more details about the architecture of our network, please see Appendix 7.4.

We assume access to ground truth safety envelopes at training time. This can be directly obtained in simulated environments or from auxiliary sensors such as LiDAR in the real world. Because a light curtain is an active sensor, the data that it collects depends on the forecasting policy. Thus to train our network, we use DAgger [RGB11], a widely-used imitation learning algorithm to train a policy with expert or ground-truth supervision across multiple timesteps. We use the Huber loss [Hub92] between the predicted and ground truth safety envelopes as our training loss. The Huber loss is designed to produce stable gradients while being robust to outliers.

(a)


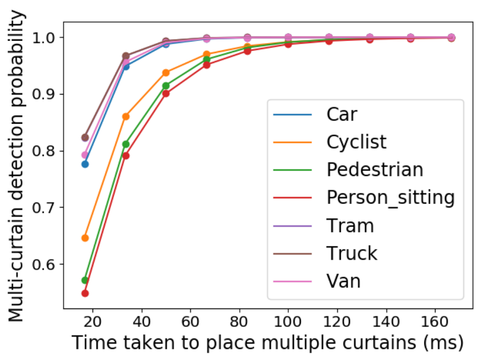
(b)


Figure 7.2: (a) The probability of random curtains detecting objects of various areas. For an object of a fixed area, we average the probability across various orientations of the object. Larger objects are detected with higher probability. (b) We show the detection probability of canonical objects from classes in the KITTI [GLU12] dataset. For each object class, we construct a "canonical object" by averaging the dimensions of all labeled instances of that class in the KITTI dataset. Larger object classes are detected with a higher probability, as expected. We also show the detection probability as a function of the number of light curtains placed. The detection probability increases exponentially with the number of light curtain placements.

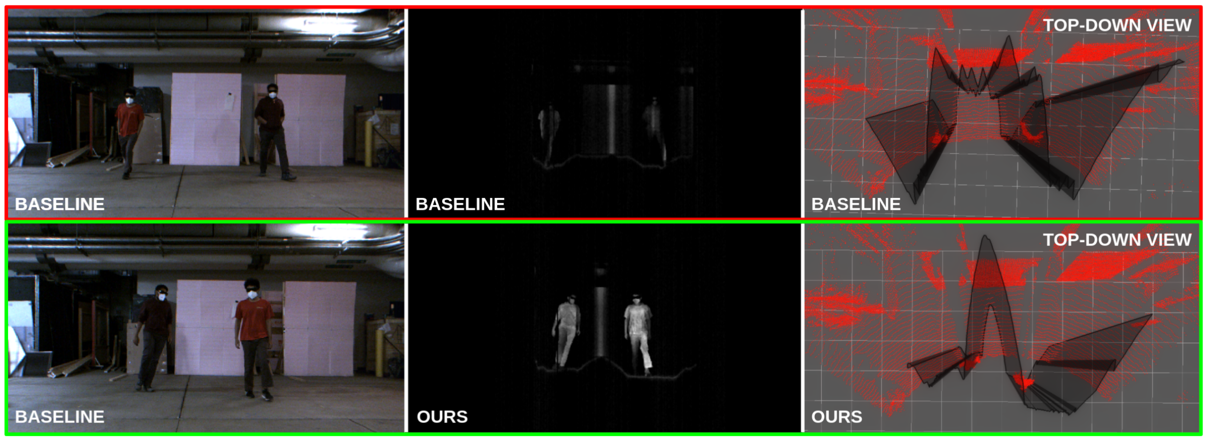


Figure 7.3: Qualitative results in a real-world environment with two walking pedestrians, comparing a hand-crafted baseline policy (top-row) with our method (bottom-row). *Left column:* contains RGB scene images. *Middle column:* contains the light curtain images, where higher intensity means a closer intersection between the light curtain and the object surfaces (i.e. a better estimation of the safety envelope). Since our method learns to forecast the safety envelope, it estimates the envelope more accurately and produces higher light curtain intensity. *Right column* (top-down view): the black surfaces are the estimated safety envelopes, and red points show a LiDAR point cloud (only used to aid visualization). Our forecasting method's estimate of the safety envelope hugs the pedestrians more tightly and looks smoother. The hand-crafted baseline works by continuously moving the curtain back and forth, creating a jagged profile and preventing it from enveloping objects tightly.

	Huber loss	RMSE Linear	RMSE Log	RMSE Log Scale-Inv.	Absolute Relative Diff.	Squared Relative Diff.	Thresh (1.25)	Thresh (1.25^2)	Thresh (1.25^3)
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
Handcrafted baseline	0.1145	1.9279	0.1522	0.0721	0.1345	1.0731	0.6847	0.7765	0.8022
Random curtain only	0.1484	2.2708	0.1953	0.0852	0.1698	1.2280	0.6066	0.7392	0.7860
1D-CNN (w/ Forecasting + Random curtains + Baseline input)	0.0896	1.7124	0.1372	0.0731	0.1101	0.7076	0.7159	0.7900	0.8138
1D-GNN (w/ Forecasting + Random curtains + Baseline input)	0.1074	1.6763	0.1377	0.0669	0.1256	0.8916	0.7081	0.7827	0.8037
Ours w/o random curtains	0.1220	2.0332	0.1724	0.0888	0.1411	0.9070	0.6752	0.7450	0.7852
Ours w/o Forecasting	0.0960	1.7495	0.1428	0.0741	0.1163	0.6815	0.7010	0.7742	0.8024
Ours w/o Baseline input	0.0949	1.8569	0.1600	0.0910	0.1148	0.7315	0.7082	0.7740	0.7967
Ours	**0.0567**	**1.4574**	**0.1146**	**0.0655**	**0.0760**	**0.3662**	**0.7419**	**0.8035**	**0.8211**

Table 7.1: Performance of safety envelope estimation on the SYNTHIA [ZBGGV+19] urban driving dataset under various metrics.

		Huber loss	RMSE Linear	RMSE Log	RMSE Log Scale-Inv.	Absolute Relative Diff.	Squared Relative Diff.	Thresh (1.25)	Thresh (1.25^2)	Thresh (1.25^3)
		$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
Slow Walking	Handcrafted baseline	0.0997	0.9908	0.1881	**0.1015**	0.1371	0.2267	0.8336	**0.9369**	**0.9760**
	Ours	**0.0630**	**0.9115**	**0.1751**	0.1083	**0.0909**	**0.1658**	**0.8660**	0.9228	0.9694
Fast Walking	Handcrafted baseline	0.1473	1.2425	0.2475	0.1508	0.1824	0.3229	0.6839	0.8774	**0.9702**
	Ours	**0.0832**	**0.9185**	**0.1870**	**0.1201**	**0.1132**	**0.2093**	**0.8575**	**0.9165**	0.9610

Table 7.2: Performance of safety envelope estimation in a real-world dataset with moving pedestrians. The environment consisted of two people walking in both back-and-forth and sideways motions.

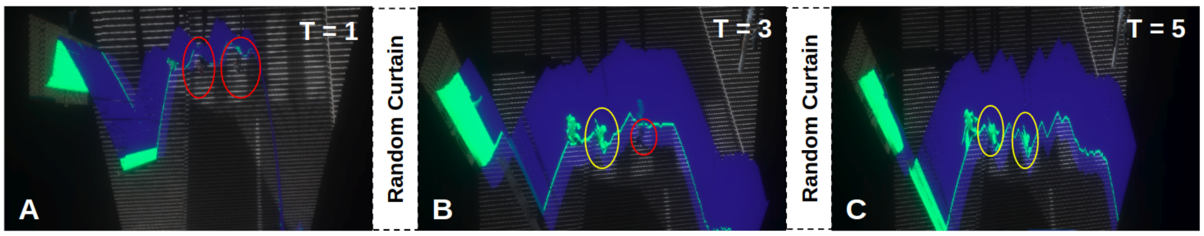


Figure 7.4: We illustrate the benefits of placing random curtains (that come with probabilistic guarantees of obstacle detection) while estimating safety envelopes, shown in SYNTHIA [ZBGGV+19], a simulated urban driving environment. The blue surfaces are the estimated safety envelopes, and the green points show regions of high light curtain intensity (higher intensity corresponds to better estimation). There are three pedestrians in the scene. (a) Our forecasting model fails to locate two pedestrians (red circles). (b) The first random curtain leads to the discovery of one pedestrian (yellow). (c) The second random curtain helps discover the other pedestrian (second yellow circle). The safety envelope of all pedestrians has now been detected.

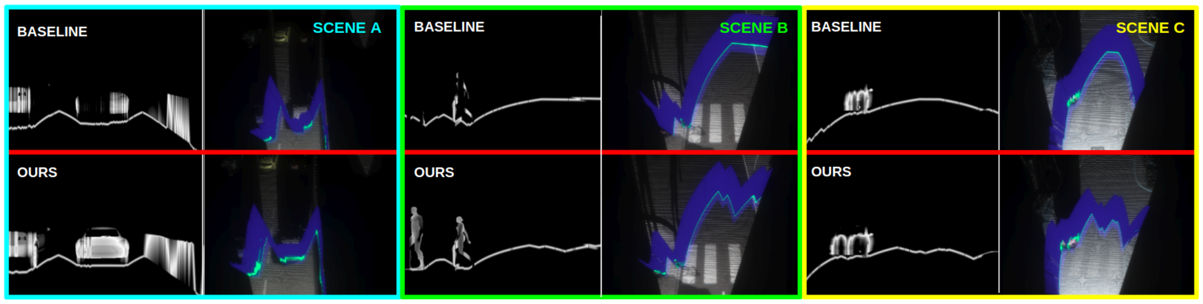


Figure 7.5: Comparison of the safety envelope estimation between a hand-crafted baseline policy (top row) and a trained neural network (bottom row), in three simulated urban driving scenes from the SYNTHIA [ZBGGV+19] dataset. For each scene and method, the left column shows the intensity image of the light curtain; higher intensities correspond to closer intersection of the light curtain and object surfaces, implying better estimation of the safety envelope. The right column shows the light curtain profile in the scene. The trained network estimates the safety envelopes more accurately than the handcrafted baseline policy.

7.3 Experiments

7.3.1 Random curtain analysis

In this section, we use the dynamic programming approach introduced in Section 4.4.2 to analyze the detection probability of random curtains. First, we compare our dynamic programming method to an alternate approach to compute detection probabilities: Monte Carlo sampling. This method involves sampling a large number of random curtains and returning the average number of curtains that were able to detect the object. This produces an unbiased estimate of the single-curtain detection probability, with a variance based on the number of samples. However, our dynamic programming approach has multiple advantages over such a sampling-based approach:

1. Dynamic programming produces an *analytic* estimate of the detection probability, whereas sampling produces a *stochastic*, noisy estimate of the probability. Analytic estimates are useful for reliably evaluating the safety and robustness of perception systems.

2. Dynamic programming is significantly more efficient than sampling based approaches. The former only involves one pass through the constraint graph. In contrast, a large number of samples may be required to provide a reasonable estimate of the detection probability.

The two methods are compared in Figure 7.1, which shows the estimated single-curtain detection probabilities of both methods as a function of the runtime of each method (the runtimes include pre-processing steps such as raycasting, and hence are directly comparable between the two methods). Dynamic programming (shown in red) produces an analytic estimate very efficiently (around 0.8 seconds). For Monte Carlo sampling, we show the probability estimate for a varying number of Monte Carlo samples. Each run shows the mean estimate of the detection probability (blue dots), as well as its corresponding 95%-confidence intervals (blue bars). Using more samples produces more accurate estimates with smaller confidence intervals, at the cost of increased runtime. The sampling approach will eventually converge to the point estimate output by dynamic programming in the limit of an infinite number of samples. This experiment shows that dynamic programming produces precise estimates (i.e. there is zero uncertainty in its estimate) while being orders of magnitude faster than Monte Carlo sampling.

Next, we investigate how the size of an object affects the detection probability. We generate objects of varying sizes and run our dynamic programming algorithm to compute their detection probabilities. Figure 7.2 (a) shows a plot of the detection probability of a single curtain as a function of the area of the object (averaged over multiple object orientations). As one would expect, the figure shows that larger objects are detected with higher probability.

Last, we analyze the detection probability as a function of the number of light curtains placed. The motivation for using multiple curtains to detect objects is the following. A single curtain might have a low detection probability p, especially for a small object. However, we could place multiple (say n) light curtains and report a successful detection if at least one of the n curtains detects the object. Then, the probability of detection increases exponentially by $1 - (1 - p)^n$. We call this the "multi-curtain" detection probability. Figure 7.2 (b) shows the multi-curtain detection probabilities for objects from the KITTI [GLU12] dataset, as a function of the time taken to place those curtains (at 60 Hz). For each object class, we construct a "canonical object" by averaging the dimensions of all labeled instances of that class in the KITTI dataset. We can see that larger object classes are detected with a higher probability, as expected. The figure also shows that the probability increases rapidly with the number of random curtains for all object classes. Four random curtains (which take about 67ms to image) are sufficient to detect objects from all classes with at least 90% probability. Note that there is a tradeoff between detection probability and runtime of multiple curtains; guaranteeing a high probability requires more time for curtains to be placed.

7.3.2 Estimating safety envelopes

Environments: In this section, we evaluate our approach to estimate safety envelopes using light curtains, in two environments. First, we use SYNTHIA [ZBGGV+19], a large, simulated dataset containing photorealistic scenes of urban driving scenarios. It consists of 191 training scenes ($\sim 96K$ frames) and 97 test scenes ($\sim 45K$) frames and provides ground truth depth maps. Second, we perform safety envelope estimation in a real-world environment with moving pedestrians. These scenes consist of two people walking in front of the device in complicated, overlapping trajectories. We perform evaluations in two settings: a *"Slow Walking"* setting, and a harder *"Fast Walking"* setting where forecasting the motion of the safety envelope is naturally more challenging. We use an Ouster OS2 128-beam LiDAR (and ground-truth depth maps for the SYNTHIA dataset) to compute ground truth safety envelopes for training and evaluation. We evaluate policies over a horizon of 50 timesteps in both environments.

Evaluation metrics: Safety envelopes can be thought of as 1D depth maps computed from a full 2D depth map, since the safety envelope is constrained to be a vertically ruled surface that always "hugs" the closest obstacle. Thus, the safety curtain can be computed by selecting the closest depth value along each column of a 2D depth map (ignoring points on the ground or above a maximal height). Because of the relationship between the safety envelope and the depth map, we evaluate our method using a variety of standard metrics from the single-frame depth estimation literature [ZSZ+20, EPF14]. The metrics are averaged over multiple timesteps to evaluate the policy's performance across time.

Baselines: In Table 7.1, we compare our method to the hand-crafted policy described in Sec. 7.2. A 'random curtain only' baseline tests the performance of random curtains for safety envelope estimation in the absence of any forecasting policy. We also compare our method against two other neural network architectures that forecast safety envelopes: a CNN that performs 1D convolutions, and a graph neural network with nodes corresponding to camera rays. Both baselines differ from our method in network architecture only; they also perform forecasting, place random curtains and use the handcrafted baseline's predictions as input. Please see Appendix 7.4 for more details about their network architectures. See Table 7.1 for a comparison of our method with the baselines in the SYNTHIA environment, and Table 7.2 for the real-world environment. The arrows below each metric in the second row denote whether a higher value (↑) or lower value (↓) is better. In both environments (simulated and real), our method outperforms the baselines on most metrics, often by a significant margin.

Ablations: We also perform multiple ablation experiments. First, we train and evaluate our without using random curtains (Tab. 7.1, "Ours w/o Random Curtains"). This reduces the performance by a significant margins, suggesting that it is crucial to combine forecasting with random curtains for increased robustness. See Appendix **??** for more experiments performed without using random curtains for all the other baselines and ablation conditions. Second, we perform an ablation in which we train our model without forecasting to the next timestep i.e. the network is only trained to predict the safety envelope of the current timestep (Tab. 7.1, "Ours w/o Forecasting"). This leads to a drop in performance, suggesting that it is important to place light curtains at the locations where the safety envelope is expected to move to, not where it currently is. Finally, we modify our method to not take the output of the hand-crafted policy as input (Tab. 7.1, "Ours w/o Baseline input"). The drop in performance shows that providing the neural network access to another policy that performs reasonably well helps with training and improves performance.

Qualitative anaysis: We perform qualitative analysis of our method in the real-world environment with moving pedestrians in Fig. 7.3, and in the SYNTHIA [ZBGGV+19] simulated environment in Figs. 7.4, 7.5. We compare our method against the hand-crafted baseline, as well as show how placing random curtains can discover objects and improve the estimation of safety envelopes. Please see captions for more details. Our project website contains videos demonstrating the qualitative performance of our method in the real-world pedestrian environment. They show that our method can generalize to multiple obstacles (as many as five pedestrians) and extremely fast and spontaneous motion, even though such examples were not part of the training set.

7.4 Network architectures and training details

In this section, we describe in detail the network architectures used by our main method, as well as various baseline models.

2D-CNN

The 2D-CNN architecture we use to forecast safety envelopes is shown in Fig. 7.6. It takes as input the previous k light curtain outputs. These consists of the intensities of the light curtain per camera ray $\mathbf{I}_{1:T}$, as well as the control points of the curtain that was placed i.e. $\mathbf{X}_{1:T}$. Each light curtain output $(\mathbf{X}_{1:T}, \mathbf{I}_{1:T})$ is converted into a *polar occupancy map*. A polar occupancy map is a $T \times L$ image, where the t-th column of the image corresponds to the camera ray R_t. Each ray is binned into L uniformly spaced locations; although L could be set to the number of control points per camera ray in the light curtain constraint graph, it is not required. Each column of the occupancy map has at-most one non-zero cell value. Given $\mathbf{X}_t, \mathbf{I}_t$, the cell on the t-th column that lies closest to $\mathbf{X}_t$ is assigned the value $\mathbf{I}_t$. We generate k such top-down polar occupancy maps encoding intensities. We generate k more such polar occupancy maps, but just assigning binary values to encode the control points of the light curtain. Finally, another polar occupancy map is generated using the forecast of the safety envelope from the handcrafted baseline policy. The $2k+1$ maps are fed as input to the 2D-CNN. We use $k=5$ in all our experiments. The input is transformed through a sequence of 2D convolutions; the convolutional layers are arranged in a manner similar to the the U-Net [RFB15] architecture. This involves skip connections between downsampled and upsampled layers with the same spatial size. The output of the U-Net is a 2D image. The U-Net is a fully convolutional architecture, and the spatial size of the output is equal to the spatial size of the input. Column-wise soft-max is then applied to transform the output into T categorical probability distributions, one per column. We sample a cell from the t-th distribution, and the location of that cell in the top-down view is interpreted as the t-th control point. This produces a forecasted safety envelope.

1D-CNN

We use a 1D-CNN as a baseline network architecture. The 1D-CNN takes as input the previous k light curtain placements $\mathbf{X}_{1:T}$, and treats it as a 3-channel 1-D image (the three channels being the x-coordinate, the z-coordinate, and the range $\sqrt{x^2+z^2}$). It also takes the previous k intensity outputs $\mathbf{I}_{1:T}$, and treats them as 1-dimensional vectors. It also takes as input the forecasted safety envelope from the hand-crafted baseline. The overall input to the 1D-CNN is a $4k+1$ channel 1D-image. It applies a series of 1D fully-convolutional operations, with ReLU activations. The output is a 1-D vector of length T. These are treated as ranges on each light curtain camera ray, and are converted to the control points $\mathbf{X}_{1:T}$ of the forecasted safety envelope. Random curtains are placed alongside this method when the forecasted envelope is being computed.

1D-GNN

We use a graph neural network as a baseline to perform safety envelope forecasting. The GNN takes as input the output of the previous two light curtain placements. The GNN contains $2T$ nodes, T nodes corresponding to each curtain. The graph contains two types of edges: vertical edges between corresponding nodes of the two curtains (T in number), and horizontal edges between nodes corresponding to adjacent rays of the same curtain ($2T-1$ in number). Each

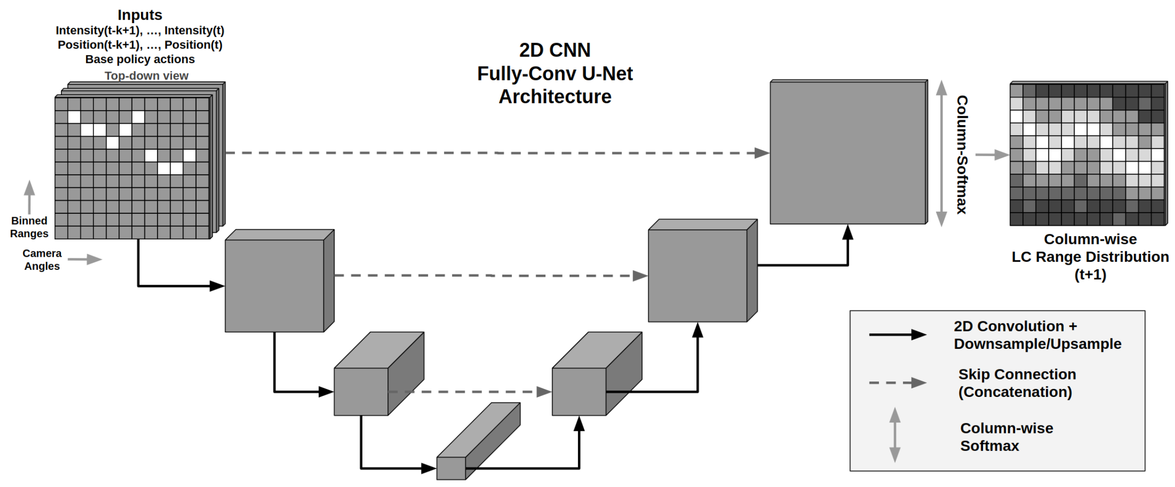


Figure 7.6: The network architecture of the 2D CNN model used for safety envelope forecasting. It takes as input the previous k light curtain outputs, and converts them into top-down polar occupancy maps. Each column of the image is assigned to a camera ray, and each row is treated as a binned location. It also takes the prediction of the hand-crafted baseline as additional input. The input is transformed through a series of 2D convolution layers, arranged in a manner similar to the U-Net [RFB15] architecture. This involves skip connections between downsampled and upsampled layers with the same spatial size. The output of the U-Net is a 2D image. This is a fully-convolutional architecture, and the spatial dimensions of the input and output are equal. Column-wise soft-max is then applied to the output to transform it to a probability distribution per column. A value $\mathbf{X}_t$ is sampled per column to produce the profile of the forecasted safety envelope.

node takes as input the intensity value of its corresponding curtain and camera ray, as well as the location of the forecasted safety envelope from the hand-crafted baseline. Each horizontal and vertical edge gets 3 input features: the differences in the $x, z, \sqrt{x^2 + z^2}$ coordinates of the control points of the rays corresponding to the nodes the edge is connected to. Then, a series of graph convolutions are applied. The features after the final graph convolution, on the nodes corresponding to the most recent light curtain placement are treated as range values on each camera ray R_t. The t-th range value is converted to a control point $\mathbf{X}_t$ for camera ray R_t, and the GNN generates a forecast $\mathbf{X}_{1:T}$ of the safety envelope. Random curtains are placed alongside this method when the forecasted envelope is being computed.

We find that providing the output of the hand-crafted baseline policy as input to the neural networks improves performance (compare the last two rows of Table 7.1). We attribute this improvement to two reasons:

1. It helps *avoid local minima during training*: when training the neural networks without the handcrafted input, we observe that the networks quickly settle into local minima where the loss is unable to decrease significantly. This suggests that the input helps with training.

2. It *provides useful information to the network*: To determine if it is also useful after training is complete, we replace the handcrafted input with a constant value and find that this significantly deteriorates performances. This indicates that the 2D CNN continues to rely on the handcrafted inputs at test time.

7.5 Parallelized pipelining and runtime analysis

In this section, we describe the runtime of our approach. Our overall pipeline has three components: (1) *forecasting* the safety envelope, (2) *imaging* the forecasted and random light curtains, and (3) *processing* the light curtain images. Since these processes can be run independently, we implement them as parallel threads that run simultaneously. This is shown in Figure 7.7.

The imaging and processing threads run continuously at all times. If a forecasted curtain is available to be imaged, it is given priority and is scheduled for the next round of imaging. But if there are no forecasted curtain waiting to be imaged, random curtains are placed and processed. This scheduling leads to an overall latency of 75ms (13.33 Hz). Due to the parallelized implementation, we are able to place two random curtains during each cycle of our pipeline.

Figure 7.7 (*right*) shows a breakdown of the timing of the forecasting method method. It consists of the feed-forward pass of the 2D CNN, as well as other high-level processing tasks.

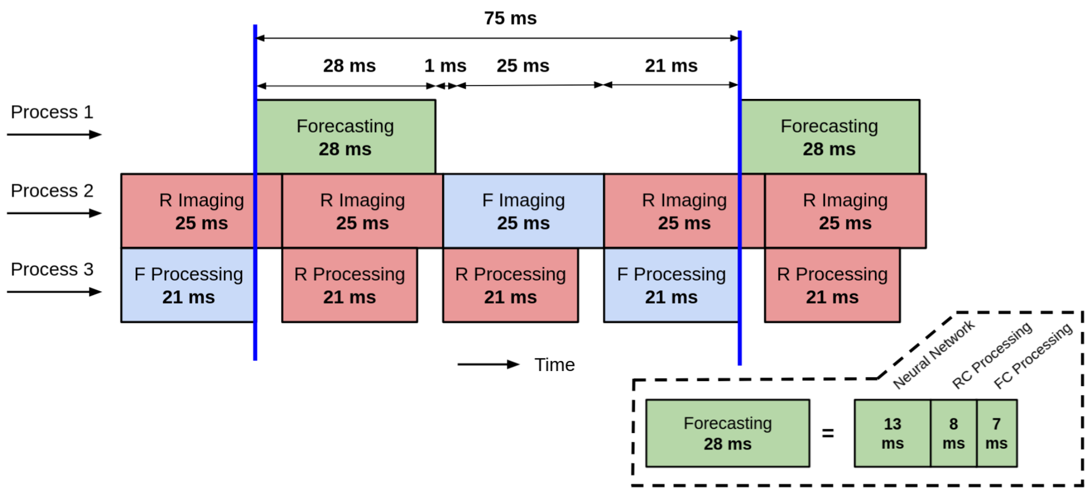


Figure 7.7: Pipeline showing the runtime of the efficient, parallelized implementation of our method. The pipeline contains three processes running in parallel: (1) the method that forecasts the safety envelope, (2) imaging of the light curtains performed by the physical light curtain device, and (3) low-level processing of images. Here, “R" or “RC" stands for “random curtain" and “F" or “FC" stands for “forecasting curtain". *Bottom right:* the forecasting method further consists of running the feed-forward pass of the 2D CNN and high-level processing of the random and forecasting curtains. The overall latency of our pipeline is 75ms (13.33 Hz). We are able to place two light curtains in each cycle of the pipeline.

7.6 Conclusion

In this work, we develop a method to estimate the safety envelope of a scene, which is a hypothetical vertical surface that separates a robot from all obstacles in the environment. We use light curtains, an actively controllable, resource-efficient sensor to directly estimate the safety envelope. We describe a method to generate random curtains that respect the physical constraints of the device, in order to quickly discover the safety envelope of an unknown object. Importantly, we develop a dynamic-programming based approach to produce theoretical safety guarantees on the probability of random curtains detecting objects in the scene. We combine this method with a machine-learning based model that forecasts the motion of already-discovered safety envelopes to efficiently track them. This enables our robot perception system to accurately estimate safety envelopes, while our probabilistic guarantees help certify its accuracy and safety towards obstacle detection and avoidance.

CHAPTER 8

Active Velocity Estimation

8.1 Introduction

The previous chapters have focused on estimating "where objects are" i.e. estimating the locations of objects using light curtains. In this chapter, we investigate estimating "how objects move". That is, our task is to estimate the *velocity* of each point in the scene. Velocity estimation is useful for the following reasons:

1. **Deciding where to place light curtains:** One of the primary reasons to perform velocity estimation is to decide where to place the light curtain in the next timestep. Velocity estimates can tell us where objects and points are expected to be in the future. Then, light curtains can be placed to sense the anticipated locations.

   In Chapter 7 Section 7.2, we learned to forecast how safety envelopes move in the next timestep. This can be thought of as *implicit* velocity estimation, where the neural network directly predicts where to place the curtain in the next timestep. However, explicitly estimating the velocities of objects and points allows them to be used for other tasks in robot perception and navigation, which we describe next.

2. **Trajectory forecasting:** Velocity estimates are a good starting point to forecast the full trajectories of objects and points into the future. The simplest forecasting method could be to output a linear motion trajectory with an assumption of constant velocity; this can be set to the predicted instantaneous velocity. More general trajectories can be predicted using a learning-based approach with the instantanous velocities as input.

3. **Obstacle avoidance and motion planning:** The instantaneous velocity (or trajectory) estimates of obstacles can be input to a motion planner [CLH+05, ZCX+21, CZCZ21]. The planner can then compute feasible robot paths that avoids obstacles. Accurate estimates of velocity would be crucial for obstacle avoidance and safety in navigation.

4. **Mapping/SLAM with dynamic objects:** In the most standard formulation of robot mapping and SLAM [Thr02, CLH+05], the objective is to compute a map of only the static objects in the scene. Applications include mapping buildings, factories and warehouses, streets and urban environments etc. This problem especially hard in the presence of dynamic objects, such as humans and vehicles moving in the scene, while the robot is mapping static objects. Heuristic methods to "subtract" dynamic objects

often fail in many situations. However, a more reliable method could be to explicitly estimate the velocity of objects, and classify them as static or dynamic. Points that are classified as dynamic can then be subtracted. A simple criterion could be to classify points as dyanmic if the magnitude of their velocity is above a certain threshold.

In this chapter, we first revisit the "dynamic occupancy grid" [DON11] approach for explicitly estimating the occupancies and velocities of locations in the scene. We provide a principled mathematical formulation to ground this approach in the well-studied framework of Bayes filtering and recursive Bayesian estimation [Thr02].

Armed with this inference method, we investigate strategies to intelligently place light curtains and improve velocity estimates. We revisit ideas from previous works [ARH+20, APNH21] in the context of dynamic occupancy grids. Furthermore, we propose a novel method to intelligently place curtains by combining the existing strategies. We develop a multi-armed bandits method to intelligently switch between multiple light curtain placement methods. Importantly, this is enabled by a novel *self-supervised* reward function that evaluates the current estimates of occupancy and velocity using future light curtain placements, without requiring ground truth or additional sensors. By reusing intermediate quantitites computed during recursive Bayes estimation of dynamic occupancy grids, we show that our self-supervised reward can be computed "for free" without requiring any extra light curtain placements or extra forecasting operations. Our multi-armed bandits method is able to outperform each of its constituent strategies when used in isolation. The algorithms developed in this chapter pave the way for controllable light curtains to accurately, efficiently, explicitly and purposefully perceive complex and dynamic environments.

8.2 Bayes filtering using dynamic occupancy grids

8.2.1 Bayes filtering

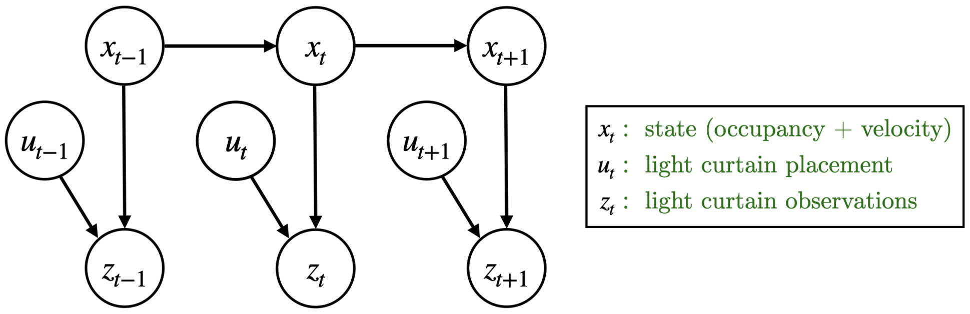

x_{t-1}
x_t
x_{t+1}
u_{t-1}
u_t
u_{t+1}
z_{t-1}
z_t
z_{t+1}
x_t : state (occupancy + velocity)
u_t : light curtain placement
z_t : light curtain observations

Figure 8.1: A *Dynamic Bayes network* [Thr02] for controllable sensing. At timestep t, x_t corresponds to the state of the world, u_t corresponds to the action i.e. the location of light curtain placement, and z_t correpsonds to light curtain measurements. $\overline{bel}(x_t)$ and $bel(x_t)$ and are the inferred distributions over states before and after incorporating measurements z_t, respectively. This is a slightly modified graphical model for controllable sensing where actions u_t dont't affect state x_t but only (and directly) affect observations z_t.

In this section, we provide a brief background on Bayes filtering. A *Dynamic Bayes filter* [Thr02], also known as a *hidden Markov model* or a *state space Model* can be represented by a probabilistic graphical model, as shown in Fig. 8.1. The state of the world at timestep t is denoted by x_t. For our problem, x_t is the occupancy and velocity of each cell in a grid. The control actions are denoted by u_t; in this work, actions are the locations where a light curtains is decided to be placed. Observations obtained from the sensor are denoted by z_t. For us, they are light curtain measurements which partially observe the occupancy of a subset of cells in the grid (more details on light curtain measurements are provided in Sec. 8.2.4).

The goal of a *Bayes filter* is to infer at each timestep t a distribution over the current state x_t from the sequence of sensor observations $z_{1:t}$ (i.e. $z_1, \dots, z_t$) and the known sequence of actions $u_{1:t}$ (i.e. $u_1, \dots, u_t$). More formally, the goal is to compute at each timestep $bel(x_t) = P(x_t \mid u_{1:t}, z_{1:t})$. The dynamic occupancy grid, like conventional occupancy grids [Elf89, Thr02], are an instance of Bayes filters. The Bayes filter systematically incorporates sensor measurements to estimate the state of the environment while also accounting for the dynamics of the world. $bel(x_t)$ is computed recursively using a process known as *recursive Bayesian estimation*. The definition of $bel(x_t)$ and the Markov property of the dyanmic Bayes network leads us to the following recursive relationship [Thr02]:

$$\begin{aligned}
bel(x_t) &= P(x_t \mid u_{1:t}, z_{1:t}) \\
&\propto P(x_t, z_t \mid u_{1:t}, z_{1:t-1}) \\
&= P(z_t \mid x_t, u_{1:t}, z_{1:t-1})\ P(x_t \mid u_{1:t}, z_{1:t-1}) \\
&= P(z_t \mid x_t, u_t)\ P(x_t \mid u_{1:t-1}, z_{1:t-1}) \\
&= P(z_t \mid x_t, u_t)\ \int_{x_{t-1}} P(x_{t-1}, x_t \mid u_{1:t-1}, z_{1:t-1})\ \mathrm{d}x_{t-1} \\
&= P(z_t \mid x_t, u_t)\ \int_{x_{t-1}} P(x_{t-1} \mid u_{1:t-1}, z_{1:t-1})\ P(x_t \mid x_{t-1}, u_{1:t-1}, z_{1:t-1})\ \mathrm{d}x_{t-1} \\
&= P(z_t \mid x_t, u_t)\ \int_{x_{t-1}} \underbrace{P(x_{t-1} \mid u_{1:t-1}, z_{1:t-1})}_{bel(x_{t-1})}\ P(x_t \mid x_{t-1})\ \mathrm{d}x_{t-1} \\
&= P(z_t \mid x_t, u_t)\ \int_{x_{t-1}} bel(x_{t-1})\ P(x_t \mid x_{t-1})\ \mathrm{d}x_{t-1} \\
&= P(z_t \mid x_t, u_t)\ \overline{bel}(x_t), \quad \text{(Measurement update step)}, \quad \text{where} \\
\overline{bel}(x_t) &= \int_{x_{t-1}} bel(x_{t-1})\ P(x_t \mid x_{t-1})\ \mathrm{d}x_{t-1} \quad \text{(Motion update step)}
\end{aligned}$$

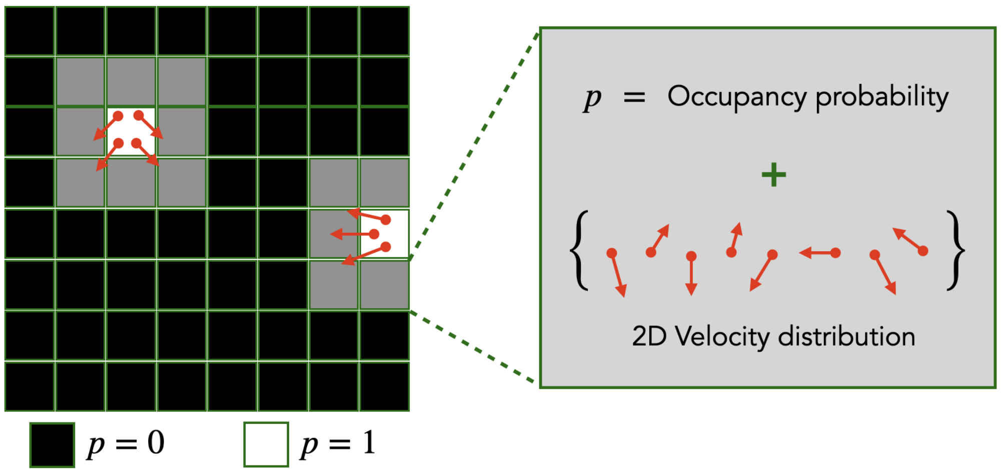
Based on the above recursive equations, recursive Bayesian estimation alternates between the following two steps:

1. **Prediction step (or motion update):** This step accounts for the dynamics of the environment. It first computes an intermediate quantity defined above: $\overline{bel}(x_t) = \int_{x_{t-1}} bel(x_{t-1})\ P(x_t \mid x_{t-1})\ \mathrm{d}x_{t-1}$. This is the result of "applying" a known or assumed motion model $P(x_t \mid x_{t-1})$ to the previous belief $bel(x_{t-1})$. In dynamic occupancy gridss, this step accounts for the motion of scene points based on their current 2D velocities. The occupancies and velocities of the next timestep are computed based on the occupancies and velocities in the previous timestep. We use a constant velocity motion model with Gaussian noise in both velocity and position. When the motion model is applied to Fig. 8.2 (a), it is updated to Fig. 8.2 (b) (illustration only). This correction step normally increases the uncertainty in occupancies and velocities.

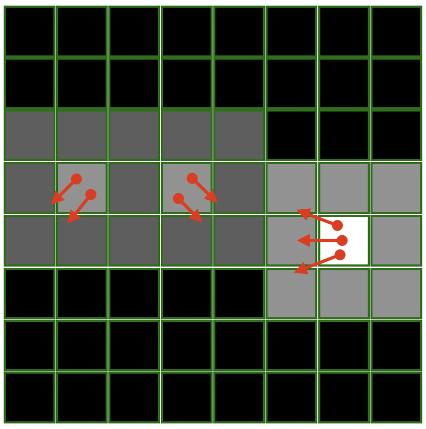
2. **Correction step (or measurement update):** This step incorporates measurements from a sensor. It updates the prior belief $\overline{bel}(x_t)$ to $bel(x_t) \propto P(z_t \mid x_t, u_t)\ \overline{bel}(x_t)$ by weighting $\overline{bel}(x_t)$ by the likelihood of the observed measurements $P(z_t \mid x_t, u_t)$. This

step often reduces uncertainty in the state. In dynamic occupancy grids, occupancies are updated using the measurements from the light curtain. Since light curtains (or any depth sensor) only measure the location of objects, this step does not update velocities. The velocity estimates are automatically refined in subsequent motion update steps. The measurement update reduced the probability of one of the positions of an object in Fig. 8.2 (b) to Fig. 8.2 (c) (illustration only).

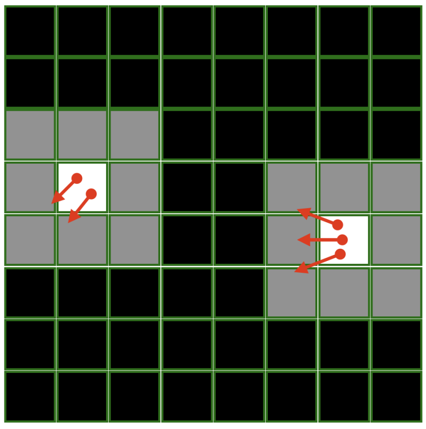
8.2.2 Particle-based dynamic occupancy grids



(a) Dynamic occupancy grid before the prediction (motion update) step.



(b) After prediction (motion update) step.



(c) After correction (measurement update) step.

Figure 8.2: *Dyanmic occupancy grid.* **Grid structure:** The 2D grid represents the top-down view and is made up of cells. Like conventional static occupancy grids [Elf89, Thr02], each cell contains an occupancy probability $p \in [0, 1]$. Dark indicates low occupancy probability and bright indicates high occupancy probability. In addition, each cell also contains a set of weighted *particles* where each particles stores a single 2D velocity. The set of particles together represents a probability distribution of that cell's velocity. **Grid updates:** The grid is a Bayes filter [Thr02] that consists of two steps: the prediction (motion update) step and the correction (measurement update) step. (a) The grid before performing any update. (b) *Prediction (motion update) step*: the occupancies and velocities of the next timestep are computed based on the grid in the previous timestep, increasing uncertainty. (c) *Correction (measurement update) step:* the occupancies are updated using the measurements from the light curtain, decreasing uncertainty. This will refine the velocity estimates.

Occupancy grids [Elf89, Thr02] are a standard tool in robotics for mapping the location of static objects in the environment. 2D occupancy grids that map objects from the top-down view are popular for mapping and SLAM in robot navigation. Each cell in the grid contains an *occupancy probability* $p \in [0, 1]$, denoting the probability of the cell being occupied by an object. *Dynamic* occupancy grids [DON11] are an extension of classical occupancy grids (see Figure 8.2(a)). Each cell in the grid contains both (1) the occupany probability $p \in [0, 1]$, as well as (2) a probability distribution over 2D velocities. The velocity distribution is represented by a set of weighted particles, where each particles stores a single 2D velocity. The set of weighted particles approximates the true velocity distribution.

Mathematical framework

Our method is built upon dyanmic occupancy grids introduced by Danescu. et. al. [DON11]. The authors describe particles as both representing a velocity distribution (i.e. weighted velocity hypotheses), as well as being “physical building blocks of the world". The former interpretation suggests that particles together represent the probability distribution of the velocity of a single physical scene point, whereas the latter suggests that each particle corresponds to its own scene point. Furthermore, the particles not only represent velocities, but their count represents the probability of occupancy. While the method is shown to be very promising, the precise role of particles and what they represent remains unclear. In this work, we provide a rigorous mathematical formulation of dynamic occupancy grids. We explicity state the assumptions made and provide a precise interpretation of particles. Our framework can be derived from three assumptions:

Assumption 1: There are no collisions

Each cell can be occupied by at most one physical scene point with a single velocity. Cells are sufficiently small that multiple objects with different velocities cannot exist (“collide”) within a cell.

This assumption is required for a single velocity of a cell to be well-defined. Assumption 1 paves the way for a straightforward interpretation particles: all particles belonging to a cell represent a probability distribution over the single velocity of that cell. Assumption 1 allows us to define the state space of occupancies and velocities.

Representing the state space: Each cell is indexed by $i \in \mathcal{I}$ from an index set $\mathcal{I}$ of all cell in the grid. At timestep t, the state of the i-th cell is denoted by $x_t^i = (o_t^i, v_t^i)$. It contains two variables. The first is a binary occupancy variable $o_t^i \in \{0, 1\}$ which denotes whether the cell is occupied or not. The second is a 2D velocity variable $v_t^i \in \mathbb{R}^2$ representing the continuous velocity of the cell (*if* it is occupied) from the top-down view. The overall state of the dynamic occupancy grid x_t is a concantenation of the states of all cells in the grid, i.e. $x_t = \{x_t^i = (o_t^i, v_t^i) \mid i \in \mathcal{I}\}$. Note that the variable v_t^i is “conditional”: it is only defined when the cell is occpied i.e. $o_t^i = 1$.

Assumption 2: Constant velocity motion model

Each scene point moves with a constant velocity, with added Gaussian noise.

Any motion can be approximated by a constant velocity motion model as long as the time inteval is sufficiently small. Therefore, this assumption is reasonable in our case since light curtains operate at very high speeds (∼60 Hz). Note that although we use the constant velocity motion model in this work following [DON11], the dynamic occupancy grid framework can still be used by swapping it with any other motion model.

Let $\epsilon_t^i \sim \mathcal{N}(0, R_\epsilon)$ and $\delta_t^i \sim \mathcal{N}(0, R_\delta)$ be the Gaussian noise in velocity and position respectively for the i-th cell in the grid at time t. And let pos^i denote the 2D location of the center of the i-th cell. The constant velocity motion model can be expressed mathematically as

$$o_{t+1}^j, v_{t+1}^j = \begin{cases} 1,\ v_t^i + \epsilon_t^i & \text{if } \exists i \in \mathcal{I} \text{ such that } o_t^i = 1 \text{ and } \mathrm{pos}^j \approx \mathrm{pos}^i + v_t^i\ \Delta t + \delta_t^i \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

In other words, a cell j will be occupied in the next timestep if and only if there exists another cell i in the previous timestep which moves to j under the constant velocity motion model. In that case, the velocity of cell j will be equal to that of cell i modulo the Gaussian noise. The equality is approximate taking into account the finite size of the cell.

Representing the belief distribution: Dynamic occupancy grids represent the current belief over velocities and occupancies of the environment. It is a probability distribution over states x_t described above. The state space is extremely large. Dynamic occupancy grids represent the belief compactly by making the following assumption:

Assumption 3: Cells are independently distributed

The probability distributions of all cells are independent of each other. This is a standard assumption made in occuapancy grids literature for computational tractability.

Each cell contains two distributions:

- Occupancy distribution (ω_t^i): o_t^i is a Bernoulli random variable over $\{0, 1\}$ with probability $\omega_t^i \in [0, 1]$.
- Velocity distribution (V_t^i): v_t^i is a random varaible over $\mathbb{R}^2$. It is represented by a set of M weighted particles $V_t^i = \{(v_t^{i,m}, p_t^{i,m}) \mid 1 \leq m \leq M\}$. The m-th particle has a velocity $v_t^{i,m}$ and weight $p_t^{i,m}$ that add up to 1 i.e. $\sum_{m=1}^M p_t^{i,m} = 1$. The larger the number of particles used, the better is the particle approximation to the true continous velocity distribution.

The velocity distribution acts like a "conditional" distribution. The probability that cell i is occupied with velocity $v_t^{i,m}$ is the product $\omega_t^i\ p_t^{i,m}$ of the probability of being occupied (ω_t^i) and the probability of having the velocity $v_t^{i,m}$ given that it is occupied ($p_t^{i,m}$). The probability of being unoccupied is simply $1 - \omega_t^i$. Since cells are assumed to be independently distributed, the probability of the entire grid is the product of probabilities of individual cells in the grid.

8.2.3 Motion update step

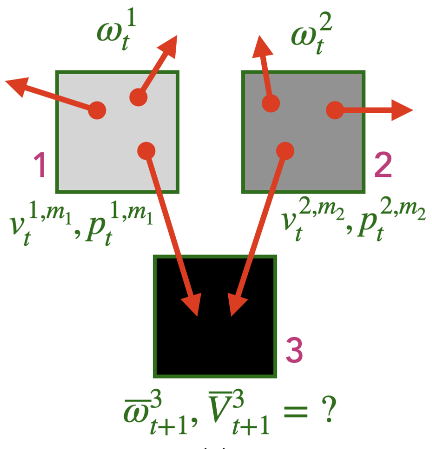
We will now derive the motion update equations of the dynamic probability grid. These equations govern how particles will move across the grid and be reweighted. Consider a simplified grid shown in Fig. 8.3 (a). Assume that there are only three cells in the grid, and only cells 1 and 2 are occupied at time t. The occupancy and velocity distributions are illustrated in the figure, and particles $(v_t^{1,m_1}, p_t^{1,m_1})$ from cell 1 and $(v_t^{2,m_2}, p_t^{2,m_2})$ from cells 2 move to cell 3 in the next timestemp $t+1$ (assuming that noise has been incorporated in v_t^{1,m_1}, v_t^{2,m_2}). What should be the occupancy and velocity distribution of cell 3?

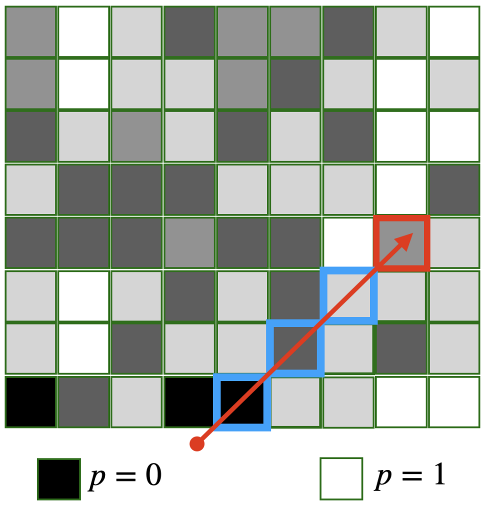
Let E_1 be the event that the particle from cell 1 enters cell 3. Similarly, let E_2 be the event that the particle from cell 2 enters cell 3. We have that $P(E_1) = \omega_t^1\ p_t^{1,m_1}$ and $P(E_2) = \omega_t^2\ p_t^{2,m_2}$. Cell 3 will be occupied when either E_1 or E_2 happens (from Eqn. 8.1). Its occupancy probability after the motion update is $\overline{\omega}_{t+1}^3 = P(E_1 \cup E_2)$. Now, from assumption 1 (the no collision assumption), objects of different velocities cannot occupy the same cell. Therefore events E_1 and E_2 must be disjoint. From the law of total probability of disjoint events, we have that $P(E_1 \cup E_2) = P(E_1) + P(E_2)$. Therefore, the occupancy probability of cell 3 after the motion update $\overline{\omega}_{t+1}^3 = \omega_t^1\ p_t^{1,m_1} + \omega_t^2\ p_t^{2,m_2}$. The conditional velocity distribution of cell 3 will comprise of v_t^{1,m_1} and v_t^{2,m_2}, with weights proportional to $P(E_1)$ and $P(E_2)$ respectively. This leads us to the general motion update of dynamic occupancy grids:

$$\overline{\omega}_{t+1}^j = \sum_{i \in \mathcal{I}} \omega_t^i \sum_{m=1}^{M_i} p_t^{i,m}\ \mathbb{I}\left[\text{pos}^j \approx \text{pos}^i + v_t^{i,m}\ \Delta t + \delta_t^{i,m}\right]$$

$$\overline{V}_{t+1}^i = \left\{ \left(v_t^{i,m} + \epsilon_t^{i,m}, \frac{\omega_t^i\ p_t^{i,m}}{\overline{\omega}_{t+1}^j} \right) \middle|\ i, m : \mathbb{I}\left[\text{pos}^j \approx \text{pos}^i + v_t^{i,m}\ \Delta t + \delta_t^{i,m}\right] \right\}$$

Note that the above derivation does not require assumption 3; the law of total probability applies even when the distributions of incoming cells are not independent! Therefore, the motion update is exact even if we treat ω_t^i and V_t^i as marginal distributions of potentially correlated cells. Assumption 3 is however required for the measurement update step (see Sec. 8.2.4).


ω_t^1
ω_t^2
1
2
v_t^{1,m_1}, p_t^{1,m_1}
v_t^{2,m_2}, p_t^{2,m_2}
3
$\overline{\omega}_{t+1}^3, \overline{V}_{t+1}^3 = ?$
(a)



(b)

Figure 8.3: (a) Applying the motion model to update occupancies and velocities of cells in the motion update step. ω's are occupancy probabilities of cells, v's and p's are velocities and weights of individual particles respectively. (b) Ray-marching procedure to compute the depth probabilities of cells along a camera ray.The depth probability of the red cell is the product of the probabilities that the red cell is occupied and that the blue cells are unoccupied.

When adding the $\omega_t^i \, p_t^{i,m}$ terms from incoming particles to compute $\overline{\omega}_{t+1}^j$, the sum should not exceed 1 under the no collision assumption (assumption 1). However, in practice, this may be violated. In such cases, we truncate the occupancy probability to 1 following [DON11]. However, this happens rarely; we needed to perform truncation only 0.35% times on average.

8.2.4 Measurement update step

Sensors such as light curtains and LiDARs provide depth information, from which the current occupancy of cells can be inferred. We use a post-processing algorithm described in [HZHR20] to process sensor data and output a detection variable $z_t^i \in \{\texttt{OCCUPIED}, \texttt{FREE}, \texttt{UNKNOWN}\}$ for each cell i in the grid. z_t^i indicates the presence, absence or lack of knowledge about objects inside cell i.

For LiDAR scans, [HZHR20] marks each cell that contains LiDAR points as OCCUPIED. Then, it uses the fact that if a 3D point was detected, light must have traveled between the sensor and the detected point in a straight line without obstruction. Therefore, rays are cast starting from the sensor to the OCCUPIED cells using an efficient voxel traversal algorithm [AW+87]. Cells lying along these rays are marked FREE. Any cells that remain unclassified are marked as UNKNOWN. This method exploits visiblity constraints of light to extract the maximum possible information from a 3D scan. We use the same processing method customized for light curtains. When a light curtain is placed on a set of cells in the grid, the cells are classified as OCCUPIED or FREE based on whether points were detected inside the cell. Raycasting to occupied cells is also performed to discover additional FREE cells. Figure 8.4 visualizes an example of visibility classification. Cells detected as OCCUPIED are shown in red. Cells shown in blue are inferred as FREE because they either lie undetected on the curtain or lie on rays cast to red cells. UNKNOWN cells are shown in gray.

The measurement update step takes the visibility classification as input. Our observation model treats this classification as a noisy observation of true occupancy. We do not update the occupancy of UNKNOWN cells. For known cells, we assume a false positive rate $\alpha_{\text{fp}} \in [0, 1]$ and a false negative rate $\alpha_{\text{fn}} \in [0, 1]$. The observation model is:

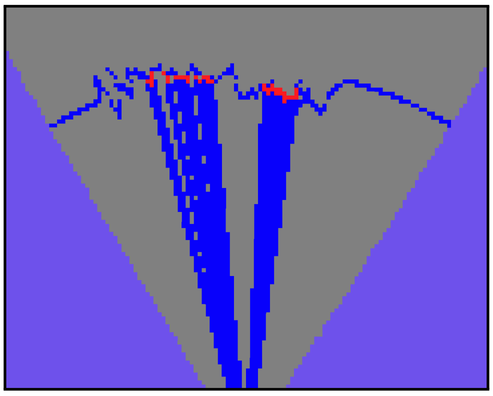


Figure 8.4: *Visibility classification from light curtain returns used in the measurement update. Red cells contain detected points and are marked as occupied. Blue cells are freespace; they either lie undetected on the light curtain or on rays cast from the sensor to the red cells. Raycasting exploits visibility constraints to extract the maximum amount of information from a light curtain. The occupancy of gray cells is unknown. Purple cells are outside the light curtain's field of view.*

$$P(z_t^i = \texttt{OCCUPIED} \mid o_t^i = 1) = 1 - \alpha_{\text{fn}}, \qquad P(z_t^i = \texttt{FREE} \mid o_t^i = 1) = \alpha_{\text{fn}}$$
$$P(z_t^i = \texttt{OCCUPIED} \mid o_t^i = 0) = \alpha_{\text{fp}}, \qquad P(z_t^i = \texttt{FREE} \mid o_t^i = 0) = 1 - \alpha_{\text{fn}}$$

We first use the assumption that all cells are independently distributed (assumption 3) to write the belief of the overall grid $\overline{bel}(x_t) = \prod_{i \in \mathcal{I}} \overline{bel}(x_t^i)$ as a product of belief distributions across all cells in the grid. Since the likelihood function $P(z_t \mid x_t) = \prod_{i \in \mathcal{I}} P(z_t^i \mid o_t^i)$ is also independent for each cell, the updated posterior belief $bel(x_t \mid z_t) \propto \overline{bel}(x_t)\ P(z_t \mid x_t)$ can be computed independently for each cell. Given the prior occupancy distribution $\overline{\omega_t^i}$ and an observation z_t^i for cell i, its occupancy distribution after the measurement update can be computed using the Bayes rule:

$$\omega_t^i = \frac{\overline{\omega_t^i}\ P(z_t^i \mid o_t^i = 1)}{\overline{\omega_t^i}\ P(z_t^i \mid o_t^i = 1) + (1 - \overline{\omega_t^i})\ P(z_t^i \mid o_t^i = 0)}$$

Depth sensors only provide information about the occupancy of cells; they do not directly measure object velocities. Velocities are inferred indirectly by a combination of measurement- and motion- update steps. The measurement update incorporates information about occupancy and the motion update infers velocities that are consistent with occupancy across timesteps in a principled probabilistic manner. Therefore, our method estimates velocities from depth measurements without requiring explicit data association across frames!

8.3 Parallelized pipeline for velocity estimation

How do we make our perception system efficient? Our pipeline has three major components: (1) light curtain sensing, (2) Bayes filtering using dynamic occupancy grids, and (3) computing intelligent curtain from the latest grid. Our insight is that the three components need not run sequentially; they can be parallelized (see Fig. 8.5) We run the three processes in parallel threads with shared memory, at their own independent speeds.

SENSE thread: imaging light curtains. The light curtain sensing thread continously places light curtains. It places “intelligent” curtains computed by our placement algorithms. However, when waiting for the next intelligent curtain placement command, instead of sitting idle, it places random curtains (that are generated offline using the constraint graph). This ensures that the light curtain device is continuously placing light curtains and is always kept busy. This allows it to place curtains at its maximum capacity of approximately 45 Hz.

THINK thread: Bayes filtering. This thread performs inference using Bayes filtering with our dynamic occupancy grid. It performs two steps in a loop: (1) the motion update, and (2) the measurement update. It inputs light curtains imaged by the SENSE thread in the measurement update. The two steps update the dynamic occupancy grid. This thread runs at approximately 35 Hz.

ACT thread: computing intelligent curtains. This thread computes the next intelligent curtain to place using the current dynamic occupancy grid. It first uses the dynamic occupancy grid and *forecasts* it to to a future time when the next intelligent curtain is expected to be imaged. Then, we apply any one of the strategies outlined in Sec. 8.4 on the forecasted grid to compute the intelligent curtain. This intelligent curtain is sent as a command to the light curtain device to be imaged.

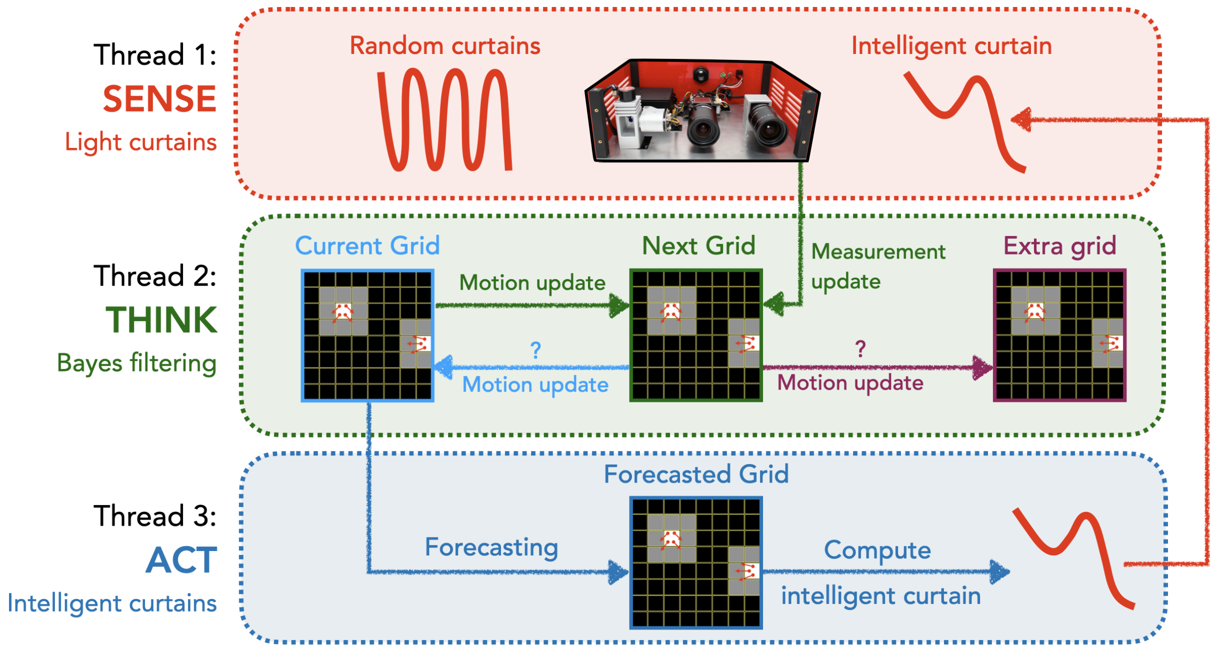


Figure 8.5: *Implementation of our method as a parallelized pipeline.* Our methods contains three components: (1) light curtain sensing, (2) inference using Bayesian filtering and (3) computing intelligent curtains. Each process can be run parallelly in a separate thread at its own independent speed. The three processes are tightly coupled in a closed loop with shared memory. Our implementation ensures that information flows between the threads safely and continuously.

Using an extra grid for thread-safety and efficiency How many grids are needed to implement parallelization, especially across the Bayes filtering and planning threads?

- The motion update step moves particles across the grid according to the motion model. The particles cannot be moved in place inside the same grid since it may cause particles to be erroneously moved more than once. Therefore, the motion update step requires two grids: a "source/current" grid and a "destination/next" grid. Particles from the current grid are copied, moved and placed in the next grid. After the motion update is complete, the roles of the current and next grids are swapped. The next grid is now assigned to be the new "current" grid since it is now the most up-to-date, incorporating the latest measurements. In the next motion update step, particles move from this grid to the older current grid (now taking the role of the "next" grid).
- The intelligent curtain thread also performs a motion update when it needs to forecast the current grid to a future timestep (when the next intelligent curtain is expected to be imaged). It uses the current grid of the Bayes filtering thread as the source, but requires a third, "forecasting" grid as a destination grid.
- Although three grids are sufficient to implement parallelization, the pipeline can be made more efficient. Specifically, consider the situation where two motion updates take place simultaneously from "current" to "next" grids in the Bayes filtering thread and "current" to "forecasting" grid in the intelligent curtain thread. Once the motion update in the Bayes filtering thread is complete, it cannot immediately perform the next motion update step. It must wait for the intelligent curtain thread to finish forecasting using the "current" grid before the next motion update dirties it. If we use an additional "extra"

grid (see Fig. 8.5), the Bayes filtering thread can use this as the destination grid for its next motion update step without needing to wait on the intelligent curtain thread to finish the latter's forecasting step.

In summary, our parallelized pipeline tightly integrates the three inter-dependent processes in a closed loop. We use a total of four grids to simultaneously guarantee the following two properties: (1) grids in use are never mistakenly overwritten, and (2) no thread ever needs to wait on another to finish processing.

8.4 Intelligent curtain placement strategies

Now that we have a dynamic occupancy grid method that can infer occupancies and velocities explicitly from light curtain measurements, the main challenge is to compute the best curtain placement from the current grid i.e. the current estimates of occupancy and velocity. The measurements from the placed curtain will be fed back to upgrade the grid, closing the loop. We adapt light curtain placement strategies based on previous work [ARH+20, APNH21] to dynamic occupancy grids, as well as propose a novel method that combines the various strategies to outperform each individual strategy. We first forecast the current occupancy to the future timestep when the next light curtain is expected to be imaged. The following strategies compute an intelligent curtain using the forecasted occupancy.

8.4.1 Maximizing depth probability

In [APNH21], the authors predict the "safety envelope" of the scene and place a light curtain at the predicted location. The safety envelope is an imaginary surfaces that "hugs" or "sticks" to objects as tightly as possible. They can be thought of as *verification* curtains – they are placed to verify whether the object locations predicted by the inference model (i.e. the THINK framework) is accurate. This strategy is motivated by the fact that light curtains only detects object surfaces when it intersects them.

Note that dyanamic occupancy grids are probabilistic models. Safety envelopes or verification curtains in a dynamic occupancy grid correpsond to locations of the highest "depth probability". The depth probability of a cell in the grid is the probability that the depth of the scene along the cell's direction is the cell's location. In other words, it is the probability that a visible surface exists in the cell i.e. the cell is occupied and all other occluding cells are empty. Once we compute the depth probability of each cell in the grid, we can place a curtain that lies on the cells with the highest depth probability.

How do we compute the depth probability in a probabilistic occupancy grid? We borrow the idea of "ray marching" from the literature on volumetric rendering [TZEM17, MST+20]. In order to reconstruct the implicit depth surface from a probabilistic volume, ray marching travels along a ray originating from the sensor and computes the probability of visibility and occlusion at each point. [TZEM17] performs this for discretized 3D grids (similar to our case) whereas NeRFs [MST+20] perform this in a continuous space using neural radiance fields. Consider an example of raymarching in Fig. 8.3 (b). Let the sequence of cells on a ray be indexed as $1, 2, \ldots, n, \ldots N$. Recall from Sec. 8.2.2 that ω_t^i is the occupancy probability of the i-th cell at timestep t. The depth probability of the n-th cell $P_t^{\mathrm{D}}(n) = \omega_t^n \prod_{i=1}^{n-1}(1-\omega_t^i)$ is product of the probabilities that the n-th cell is occupied (ω_t^n) and the probabilities that each i-th cell on the ray before the n-th cell is unoccupied $(1-\omega_t^i)$ so that light can reach the n-th cell unoccluded. Let us define the "visibility" probability $P_t^{\mathrm{V}}(n) = \prod_{i=1}^{n}(1-\omega_t^i)$ that all cells are visible upto the n-th cell. Then, we have the following recursive equations:

$$\begin{aligned} P_t^{\mathrm{V}}(i) &= P_t^{\mathrm{V}}(i-1)\ (1-\omega_t^i) \\ P_t^{\mathrm{D}}(i) &= P_t^{\mathrm{V}}(i-1)\ \omega_t^i \end{aligned}$$

These recursive equations can be used to compute the depth probability of each cell along a ray efficiently in time $O(N)$ linear in the number of cells on that ray. In this strategy, we first compute the depth probability of all cells in the grid, and place a curtain at the location of the maximum depth probability along each ray.

8.4.2 Maximizing information gain

Another strategy that was found useful for 3D object detection using light curtains in [ARH+20] was to place curtains at the regions of "highest uncertainty". This is based on the principle of maximizing mutual information for active sensing.

Consider the dynamic Bayes network shown in Fig. 8.1. Given a forecasted prior belief $\overline{bel}(x_t)$, the information gain framework prescribes that the action u_t should be taken that maximizes the information gain $\mathrm{IG}(x_t, z_t \mid u_t)$ between the state x_t and the observations z_t when taking an action u_t. Information gain is a well-studied quantity in information theory and is usually defined as:

$$\mathrm{IG}(x_t, z_t \mid u_t) = \underbrace{\mathrm{H}(P(x_t))}_{\text{entropy of } x_t} - \underbrace{\mathbb{E}_{z_t|u_t}\Big[\mathrm{H}(P(x_t \mid z_t, u_t))\Big]}_{\text{conditional entropy of } x_t \mid z_t \text{ under } u_t} \tag{8.2}$$

The information gain is the expected reduction in entropy (i.e. uncertainty) in x_t before and after taking the action u_t. [ARH+20] showed that under certain assumptions, the information gain of conventional occupancy grids on placing light curtains is equal to the sum of binary entropies of the occupancy probabilities of the cells that the curtain lies on.

While computing information gain for conventional occupancy grids is straightforward to compute, it is not so for the case of dynamic occupancy grids. This is because the underlying state space of dynamic occupancy grids is a 'mixture' of discrete and continuous spaces. Consider the state x_t^i of the i-th cell in the grid. The space of the state x_t^i is:

$$x_t^i \in \underbrace{\{\text{unoccupied}\}}_{\text{discrete space}} \cup \underbrace{\{\text{occupied with } v_t^i \mid v_t^i \in \mathbb{R}^2\}}_{\text{continuous space}}$$

The cell can either be unoccupied, or be occupied with a continuous velocity. Unfortunately, the entropy of such mixed discrete-continous spaces is not well-defined [GKOV17]. Therefore the "2H-estimator" in Eqn. 8.2 cannot be used to compute information gain since the individual terms on the right hand side are not well-defined.

Fortunately, information gain is well-defined for most distributions, including discrete-continuous mixtures [GKOV17]. This is possible by using a more general definition of information gain given by the "Radon–Nikodym" derivative [GKOV17]:

$$\mathrm{IG}(x_t, z_t \mid u_t) = \int_{x_t, z_t} \underbrace{\log \frac{dP_{x,z}}{dP_x P_z}}_{\text{Radon–Nikodym derivative}} dP_{x,z} \tag{8.3}$$

The Radon–Nikodym is well-defined for discrete-continuous mixtures [GKOV17]. When the individual entropy terms of Eqn. 8.2 are well-defined, the more general definition of Eqn. 8.3 reduces to Eqn. 8.2. The two definitions are consistent.

We will now derive the information gain of dynamic occupancy grids using the more general Radon–Nikodym definition. Here, we derive the information gain of a single i-th cell. Let ω be the occupancy probability of the cell. Let the continuous velocity distribution of the cell be denoted by $P(v)$. Assume that we place a curtain on this cell, and we obtain an observation $z_t^i \in \{0, 1\}$ to be a noisy measurement of the cell's occupancy. This is assuming that we are using a depth sensor that can only partially observe occupancy but cannot directly observe velocities. Let α_{fp} and α_{fn} be the false-positive and false-negative rates of the sensor respectively. Then,

$$\begin{aligned}
\mathrm{IG}(x_t^i \mid z_t^i) &= \int_{x,z} dP_{x,z} \, \log \frac{dP_{x,z}}{dP_x P_z} \quad \text{(Radon-Nikodym formulation)} \\
&= \underbrace{(1-\omega)(1-\alpha_{\text{fp}}) \log \frac{(1-\omega)(1-\alpha_{\text{fp}})}{(1-\omega)P(z_t^i = 0)}}_{\text{unoccupied and undetected}} + \underbrace{(1-\omega)\ \alpha_{\text{fp}} \log \frac{(1-\omega)\ \alpha_{\text{fp}}}{(1-\omega)P(z_t^i = 1)}}_{\text{unoccupied and detected}} \\
&+ \int_v \underbrace{\omega\ P(v)dv\ (1-\alpha_{\text{fn}}) \log \frac{\omega\ P(v)dv\ (1-\alpha_{\text{fn}})}{\omega\ P(v)dv\ P(z_t^i = 1)}}_{\text{velocity v and detected}} + \int_v \underbrace{o\ P(v)dv\ \alpha_{\text{fn}} \log \frac{\omega\ P(v)dv\ \alpha_{\text{fn}}}{\omega\ P(v)dv\ P(z_t^i = 0)}}_{\text{velocity v and undetected}} \\
&= (1-\omega)(1-\alpha_{\text{fp}}) \log \frac{(1-\alpha_{\text{fp}})}{P(z_t^i = 0)} + (1-\omega)\ \alpha_{\text{fp}} \log \frac{\alpha_{\text{fp}}}{P(z_t^i = 1)} \\
&\quad + \omega\ (1-\alpha_{\text{fn}}) \log \frac{(1-\alpha_{\text{fn}})}{P(z_t^i = 1)} + \omega\ \alpha_{\text{fn}} \log \frac{\alpha_{\text{fn}}}{P(z_t^i = 0)} \\
&= -\Big[(1-\omega)(1-\alpha_{\text{fp}}) + \omega\ \alpha_{\text{fn}}\Big] \log P(z_t^i = 0) - \Big[(1-\omega)\ \alpha_{\text{fp}} + \omega(1-\alpha_{\text{fn}})\Big] \log P(z_t^i = 1) \\
&\quad - \omega\ \mathrm{H}(\alpha_{\text{fn}}) - (1-\omega)\ \mathrm{H}(\alpha_{\text{fp}}) \\
&= -P(z_t^i = 0) \log P(z_t^i = 0) - P(z_t^i = 1) \log P(z_t^i = 1) - \omega\ \mathrm{H}(\alpha_{\text{fn}}) - (1-\omega)\ \mathrm{H}(\alpha_{\text{fp}}) \\
&= \mathrm{H}(z) - \omega\ \mathrm{H}(\alpha_{\text{fn}}) - (1-\omega)\ \mathrm{H}(\alpha_{\text{fp}}) \\
&= \mathrm{H}(\omega) \quad \text{(assuming that } \alpha_{\text{fp}} = \alpha_{\text{fn}} = 0)
\end{aligned}$$

In the limiting case that the sensor is accurate (i.e. the false positive rate α_{fp} and the false negative rate α_{fn} are both close to zero), the information gain of a single cell due to placing a light curtain on that cell is equal to its binary entropy $\mathrm{H}(\omega) = -\omega\ \log_2 \omega - (1-\omega)\ \log_2(1-\omega)$. Therefore, the total information gain is the sum of binary cross entropies of the cells that the curtain lies on. This is similar to the information gain in [ARH+20]. However, we have been able to show this mathematically in the more complex case of mixed discrete-continuous distributions.

This theoretical intuitively makes sense. Since the depth sensor measurements only provide information about occupancy and not velocity, it is not surprising that the information gain is equal to the total occupancy uncertainty.

Occupancy uncertainty/entropy: The previous section (Sec. 8.4.2) provides an intelligent curtain placement strategy: place the curtain that maximizes the sum of binary cross entropies $\mathrm{H}_{\text{occ}}(\omega_t^i) = -\omega_t^i\ \log_2 \omega_t^i - (1-\omega_t^i)\ \log_2(1-\omega_t^i)$. We also propose two more strategies below.

Velocity uncertainty/entropy: Each cell, apart from an occupancy probability, also contains a velocity distribution $P(V_t^i) \sim \{(v_t^{i,m}, p_t^{i,m}) \mid 1 \leq m \leq M\}$ represented by a set of M weighted particles with velocities $v_t^{i,m}$ and weights $p_t^{i,m}$ that sum to one. Another strategy could be to place curtains that maximize the sum of velocity entropies.

We use a discrete set of particles to represent the velocity distribution. However, the velocity sample space and the underlying true velocity distribution is continuous. Therefore, before computing the *differential* entropy of the continuous velocity distribution, we must first estimate its continuous probability density. This can be achieved by fitting a family of continous distributions to the set of weighted particles, and computing the differential entropy of the fitted distribution. We choose the simplest of such families: the parametric family of multivariate Gaussians. However, one could pick more complex families, such as non-parametric kernel density estimators (KDE). We fit a multivariate Gaussian to the particles to obtain the mean $\mu = \sum_{m=1}^{M} p_t^{i,m}\ v_t^{i,m}$ and the covariance matrix $\Sigma = \sum_{m=1}^{M} p_t^{i,m}\ (v_t^{i,m} - \mu)\ (v_t^{i,m} - \mu)^T$. Then, we compute the differential entropy of the fitted Gaussian distribution $\mathrm{H}_{\text{vel}}(V_t^i) = \frac{1}{2} \log \det(2\pi e \Sigma)$. Finally, we place a curtain that maximizes the sum of velocity entropies, as computed above, of the cells the curtain lies on.

Combined occupancy and velocity uncertainty/entropy: We also consider maximizing a weighted combination of occupancy and velocity entropies:
$\mathrm{H}_{\text{cmb}}(\omega_t^i, V_t^i) = \mathrm{H}_{\text{occ}}(\omega_t^i) + \omega_t^i\ \mathrm{H}_{\text{vel}}(V_t^i)$. The velocity uncertainty is weighted by the occupancy probability. This captures the notion that if the occupancy probability is very low, then the overall uncertainty should also be low even if the uncertainty in velocity is high. This can be thought of as a heuristic placement policy; in practice, this policy performs very well.

8.5 Self-supervised multi-armed bandits

8.5.1 Multi-armed bandit framework

Multi-armed bandits [SB18] are an online learning framework. A multi-armed bandit consists of a set of actions or "arms", where each action is associated with an unknown reward function. The set of actions is denoted by A. The agent only observe samples from the reward distribution when it takes that action. The goal is to maximize the cumulative reward over time. It mantains a running average of the rewards for each action, called *Q-values*.

We use ϵ-greedy multi-armed bandits, that trades-off exploration with exploitation. With with probability ϵ, the bandit performs exploration and choses one among the $|A|$ actions with equal probability. With probability $1-\epsilon$, it performs explotation and chooses the action that has the highest Q-value.

We use multi-armed bandits to intelligently switch between the four curtain placement strategies described in Sec. 8.4.

8.5.2 Self-supervised rewards

The bandit framework requires a reward function to evaluate actions. How should we design a reward function that is available at test-time using only light curtain placements and measurements?

We propose to obtain the reward function in a self-supervised manner. Since our eventual goal is accurately estimating occupancy and velocity, we aim to place curtains that lead to the largest improvements in occupancy and velocity estimates. Furthermore, since we use forecasted occupancy as our evaluation metric, we will use the same for our reward function.

How do we compute the accuracy of forecasted occupancy at test time, without needing ground truth, in a self-supervised way? Let us revisit the dynamic Bayes network show in Fig. 8.6 to answer this question.

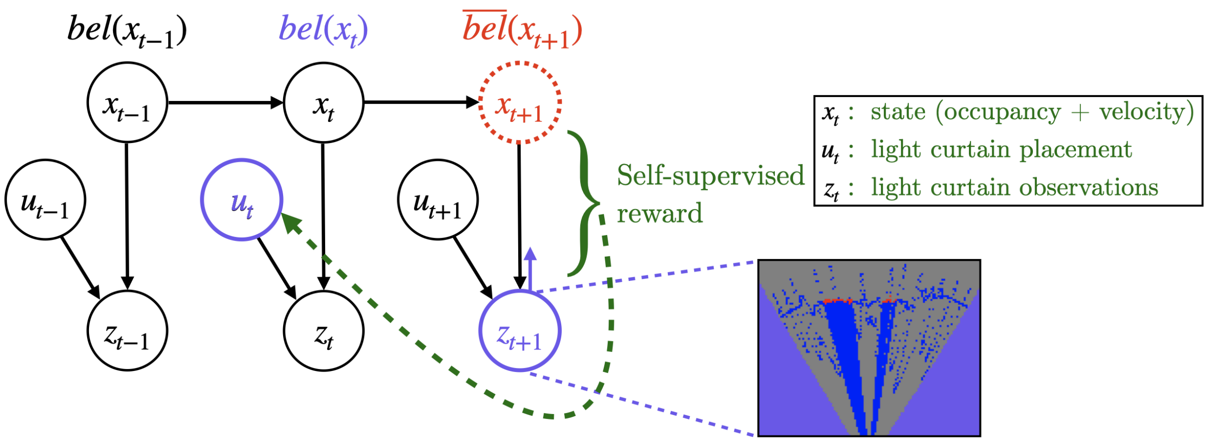

$bel(x_{t-1})$
$bel(x_t)$
$\overline{bel}(x_{t+1})$
x_{t-1}
x_t
x_{t+1}
u_{t-1}
u_t
u_{t+1}
z_{t-1}
z_t
z_{t+1}
Self-supervised reward
x_t : state (occupancy + velocity)
u_t : light curtain placement
z_t : light curtain observations

Figure 8.6: The *Dynamic Bayes network* [Thr02] from Fig. 8.1 demonstrating how self-supervised rewards are computed. Belief distributions are represented by dynamic occupancy grids. At timestep $t-1$, the belief $bel(x_{t-1})$ was forecasted by applying the motion model to obtain the prior belief $\overline{bel}(x_t)$ at timestep t. Then, the current light curtain measurement z_t obtained by placing a curtain at locations u_t is used to update this grid to $bel(x_t)$. Therefore, we attribute the accuracy of occupancy and velocity estimates of $bel(x_t)$ (evaluated using *forecasted occupancy*) to u_t. Note that the forecasted occupancy is computed in the next motion update as $\overline{bel}(x_{t+1})$. Our main insight is that the occupancy (partially) observed by the next light curtain measurements z_{t+1} in the next timestep can be used to evaluate $\overline{bel}(x_{t+1})$. This acts as a self-supervised reward for u_t. The reward is used by multi-armed bandits to intelligently switch between multiple placement policies. The self-supervised reward is computed "for free"; by reusing quantities already computed during recursive Bayes filtering, we do not require any extra forecasting steps or any extra light curtain placements.

Recall that belief distributions are represented by dynamic occupancy grids. At timestep $t-1$, the grid representing the belief $bel(x_{t-1})$ was forecasted by applying the motion model to obtain the prior belief $\overline{bel}(x_t)$ at timestep t. Then, in the measurement update step, the current light curtain measurement z_t obtained by placing a curtain at locations u_t is used to update the grid to $bel(x_t)$. Therefore, we attribute the accuracy of occupancy and velocity estimates of $bel(x_t)$ to u_t.

The accuracy of occupancy and velocity is evaluated using *forecasted occupancy*. Our task reduces to computing the accuracy of forecasted occupancy of $bel(x_t)$ in a self-supervised way. This is possible by resuing intermediate quantities output during recursive Bayesian updates.

First, note that the forecasted occupancy of $bel(x_t)$ is $\overline{bel}(x_{t+1})$ computed by the next motion update step. Our main insight is that before applying the next measurement update step, $\overline{bel}(x_{t+1})$ can be evaluated using the occupancy (partially) observed by the next light curtain measurements z_{t+1} at $t+1$. This acts as a self-supervised reward for the previous light curtain placement u_t.

The next observations z_t contain partial, but substantial information about occupancy. This information comes from multiple sources: (1) the intelligent curtain z_{t+1}, (2) (possibly multiple) random curtains that are continuously placed by the SENSE thread (see Sec. 8.3), and (3) freespace information obtained using raycasting (see Sec. 8.2.4).

An advantage of our self-supervised reward is that it is essentially computed for free! This is because (1) occupancy forecasting of $bel(x_t)$ is performed anyway as part of the motion update step, and (2) the partial occupancy information from z_{t+1} is computed anyway in the next measurement update step. By reusing quantities already computed during recursive Bayes filtering, our self-supervised reward does not require any extra forecasting steps nor any extra light curtain placements.

8.5.3 Non-stationary rewards

Vanilla multi-armed bandits assume that the reward distribution for each action is stationary. The Q-value of an action after it was been performed n times is computed as $Q_n = \frac{1}{n}\sum_{i=1}^{n} R_i$, where R_i is the reward obtained in the i-th trial. This is equivalent to the following recursive update rule: $Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$, where the Q-value is incremented by the error scaled by a decaying factor $\frac{1}{n}$.

However, in our case, a single placement strategy may not be superior to the rest at all times and in all situations. The reward distribution for each strategy (action) may change with time. Hence, we assume that our rewards are non-stationary. For non-stationary rewards, we wish to give more weight to recent rewards than to older rewards. Therefore, the decaying parameter is replaced by a constant step-size parameter α: $Q_{n+1} = Q_n + \alpha\,[R_n - Q_n]$. This weights newer rewards exponentially more than older rewards according to the expression: $Q_{n+1} = (1-\alpha)^n\, Q_1 + \sum_{i=2}^{n} \alpha\,(1-\alpha)^{n-i}\, Q_i$.

8.6 Experiments

8.6.1 Environments

Simulation environment: We use a simulated environment consisting of various blocks moving in a variety of motions (see Fig. 8.7 (a)). The environment contains cylinders and cuboids, moving in (1) linear, harmonic (oscillatory) motion along different directions, (2) curved sinusoidal motion, and (3) random Brownian motion. Since ground truth is available in simulation, we evaluate forecasted occupancy in this environment using ground truth occupancy.

Real-world environment: Our real-world environment consists of two pedestrians walking in front of the sensor in multiple directions and at different speeds (see Fig. 8.7 (b)). Since ground truth is not available in the real-world, we evaluate forecasted occupancy in this environment using partially observed occupancy from light curtain measurements (see our self-supervised reward in Sec. 8.5).

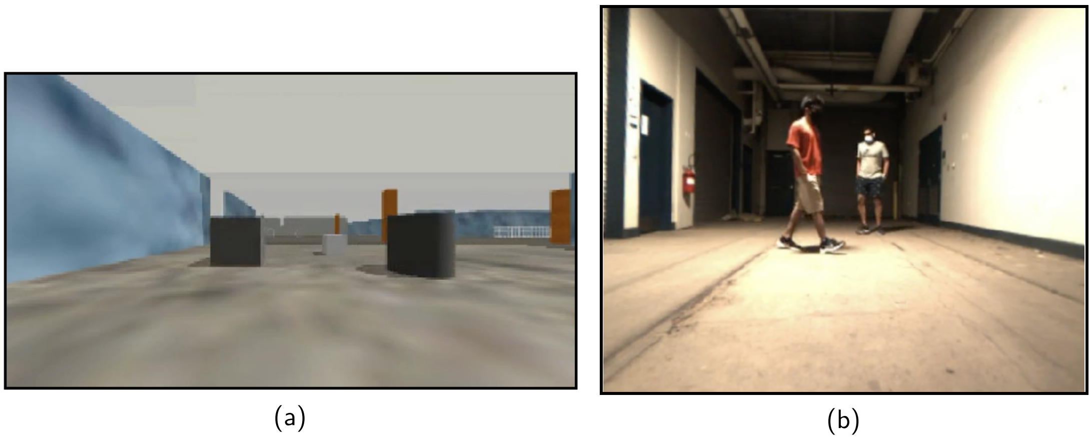


Figure 8.7: (a) *Simulated environment:* the simulated environment used for velocity estimation. It consists of differently shaped objects (cuboids and cylinders) moving in (1) linear oscillatory/harmonic motion along various directions, (2) curved sinusoidal motion, and (3) random Brownian motion. (b) *Real-world environment:* the real-world environment consists of two pedestrians walking in front of the sensor in multiple directions and at different speeds.

	Classification accuracy	Precision	Recall	F1-score	IOU
	↑	↑	↑	↑	↑
LiDAR (5 Hz)	0.9584	0.2498	0.1073	0.1360	0.0787
Random curtains only	0.9662	0.2698	0.0567	0.0850	0.0468
Max. depth probability	0.9610	0.2448	0.1146	0.1388	0.0792
Max. occupancy information gain	0.9609	0.2717	0.1266	0.1493	0.0857
Max. velocity information gain	**0.9648**	0.2728	0.0581	0.0838	0.0458
Max. occupancy + velocity information gain	0.9629	**0.3026**	0.1251	0.1544	0.0895
Multi-armed bandits (**Ours**)	0.9623	0.2814	**0.1402**	**0.1690**	**0.0976**

Table 8.1: Accuracy of occupancy and velocity estimation (measured using *forecasted occupancy* and fully observed ground truth occupancy) in simulated environments.

The velocities of the objects in both environments change continuously with time. Our task is to estimate the occupancy and velocity of each visible point on the objects.

8.6.2 Quantitative analysis

Table 8.1 and 8.2 show the performance of various light curtain placement policies under multiple metrics that evaluate forecasted occupancy. The main metrics we focus on are F1-score and IOU, shown in blue. In both sets of experiments, multi-armed bandits that combine the four other intelligent curtain placement policies using our self-supervised reward outpeform all other methods. This shows that intelligently switching between multiple placement strategies

	Classification accuracy	Precision	Recall	F1-score	IOU
	↑	↑	↑	↑	↑
Random curtains only	0.9852	0.6306	0.2357	0.2405	0.2136
Max. depth probability	0.9832	0.5943	0.2727	0.3047	0.2353
Max. occupancy information gain	0.9811	0.5733	0.3041	0.3319	0.2515
Max. velocity information gain	**0.9864**	0.6221	0.2615	0.2545	0.2232
Max. occupancy + velocity information gain	0.9822	0.5899	0.3175	0.3421	0.2727
Multi-armed bandits (**Ours**)	0.9854	**0.6467**	**0.3647**	**0.3703**	**0.3053**

Table 8.2: Accuracy of occupancy and velocity estimation (measured using *forecasted occupancy* and partially-observed, self-supervised occupancy) in a real-world environemnt with walking pedestrians.

may be more beneficial than only using one given strategy at all times.

Between the other four strategies, maximizing occupancy uncertainty and maximizing a linear combination of occupancy and velocity uncertainty perform comparably. This is followed by maximizing depth probability. Maximizing velocity uncertainty tends to perform the worst, and by a large margin in the simulated environment. Fortunately, multi-armed bandits learn to downweight the velocity uncertainty based policy (see Sec. 8.6.3). We also compare against other baselines: using only random curtains (without placing any intelligent curtains), and with a simulated Velodyne LiDAR. Unsurprisingly, only using random curtains performs the worst. All intelligent curtain policies except maximizing velocity uncertainty are able to outperform LiDAR. Interestingly, the latter policy performs comparibly to and sometimes worse than random curtains-only.

Table 8.3 shows an analysis specific to the multi-armed bandit method. The first column shows the frequencies (in percentages) of selecting each action (curtain placement strategy). The second column shows the average Q-value computed by the multi-armed bandit for each action. Higher Q-values are better; the action with the highest Q-value will be selected during exploitation. We see that the best performing policies according to Tables 8.1, 8.2 are selected most frequently. Unsurprisingly, they also have the highest Q-values. The following trend holds: that the lower the performance of an individual policy when used in isolation, the lower is its average Q-value and the less frequently it is chosen. However, a combination of all policies (MAB) is better than any one.

8.6.3 Qualitative analysis

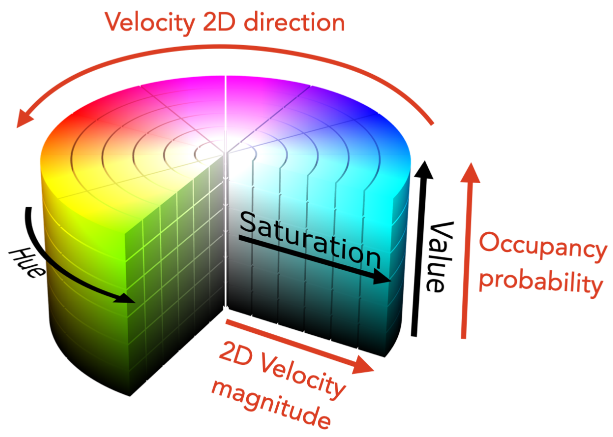
Visualizing velocities and occupancies: Fig. 8.8 shows how we visualize 2D velocities and occupancies (both ground truth and estimated velocities and occupancies). The visualization of an example ground truth grid is shown in Fig. 8.8 (b). We use the HSV colorwheel shown in Fig. 8.8 (a) to jointly visualize velocities and occupancies. The 'value' encodes the occupancy probability; dark means low occupancy probability and bright means high occupancy probability.

	Frequency of selection	Avg. Q-value function (IOU)
Max. depth probability	22.9%	0.261
Max. occupancy information gain	31.1%	0.276
Max. velocity information gain	13.4%	0.202
Max. occupancy + velocity information gain	32.5%	0.292

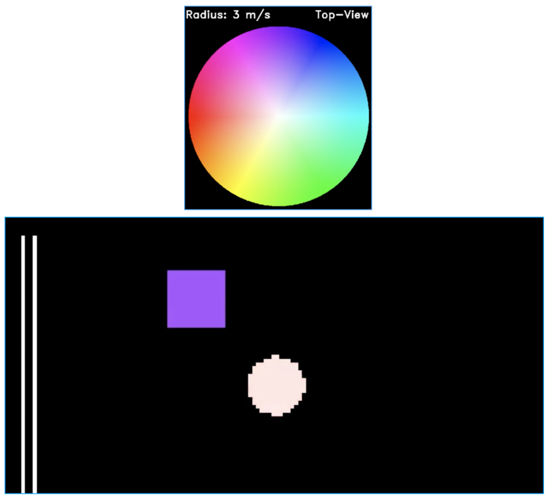
Table 8.3: Quantitative analysis of the multi-armed bandit method. The first column shows the percentage of times each action (i.e. curtain placement policy) was chosen. The second column shows the average Q-value of each action computed by the multi-armed bandit. Higher Q-value is better; the action with the highest value is selected during exploitation.

The 'hue' encodes the direction of velocity. For example, the bluish-purple hue of the cuboid in Fig. 8.8 (b) means that the cuboid is moving upwards in 2D i.e. away from the sensor in 3D. 'Saturation' encodes the magnitude of velocity. This means that white is stationary (e.g. the walls of the environment) whereas colorful corresponds to high speed.

Example from simulation. Fig. 8.10 shows an example of velocity estimation in the simulated environment. The top-left panel shows the dynamic occupancy grid with estimated velocities and occupancies. The top-right panel shows the ground truth velocities and occupancies queried from the simulator. The top-center panel shows the color-coding for 2D velocity from the top-down view. The bottom-left panel shows partial occupancies observed from light curtain measurements. This is input to the dynamic occupancy grid during the measurement



(a) HSV colorwheel used to visualize velocities.



(b) GT velocities of the simulated environment.

Figure 8.8: (a) The HSV (hue-saturation-value) colorwheel used to visualize 2D velocities and occupancies. Value corresponds to the occupancy probability. The hue corresponds to the direction of the velocity. Saturation corresponds the magnitude of velocity. (b) Ground truth velocities and occupancies of the simulated environment. The grid shows a stationary wall to the left and two objects. The bluish-purple square is moving upwards in 2D (i.e. farther away from the sensor in 3D) whereas the other objects in white are stationary.

update step. The bottom-center panel shows intelligent curtains in blue. The bottom-right panel shows the raw light curtain image. Intensities are high in the raw image when object surfaces intersect the curtain.

When objects move down, they are estimated as yellowish-green and when they move up, they are estimated as bluish-purple. Similarly, when objects move right, they are colored with a blue tinge, and when they move left, they appear reddish. The estimated velocities appear to be consistent with the ground truth.

Real-world examples. Fig. 8.11 shows qualitative results on the real-world environment using multi-armed bandits (MAB). In both figures (a) and (b), the top-right panel shows the current strategy selected by the MAB, while the bottom-right panel plots the Q-values of each action across time. Higher Q-value is better. The action with the highest Q-value will be selected during exploitation. Fig. 8.11 (a) contains two pedestrians walking at relaxed speeds. The velocity direction is correctly inferred for each pedestrian by the dynamic occupancy grid – the pedestrian walking to the right is shown in greenish-blue and the person walking to the left is colored in red. The current action selected maximizes occupancy probability. Fig. 8.11 (b) contains a harder environment where a lone pedestrain performs fast motion: running and jumping. The direction of velocity is correctly inferred as moving top-left (left and away from the sensor) i.e. reddish pink. The color saturation is high indicating that the pedestrian is indeed moving at a high speed. The current action selected maximizes depth probability. We can also see from both (a) and (b) that the MAB learns a low Q-value for the yellow policy. The yellow policy corresponds to maximizing velocity uncertainty, and is the worst performing of all 4 strategies (see Tables 8.1, 8.2). This indicates that multi-armed bandits are successfully able ignore strategies that do not perform well.

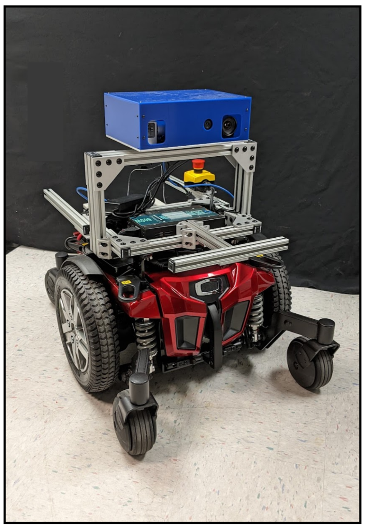


Figure 8.9: *Mobile active light curtain perception platform:* A light curtain device (in blue) is mounted on top of a mobile robot. We use this setup to perform real-world experiments.

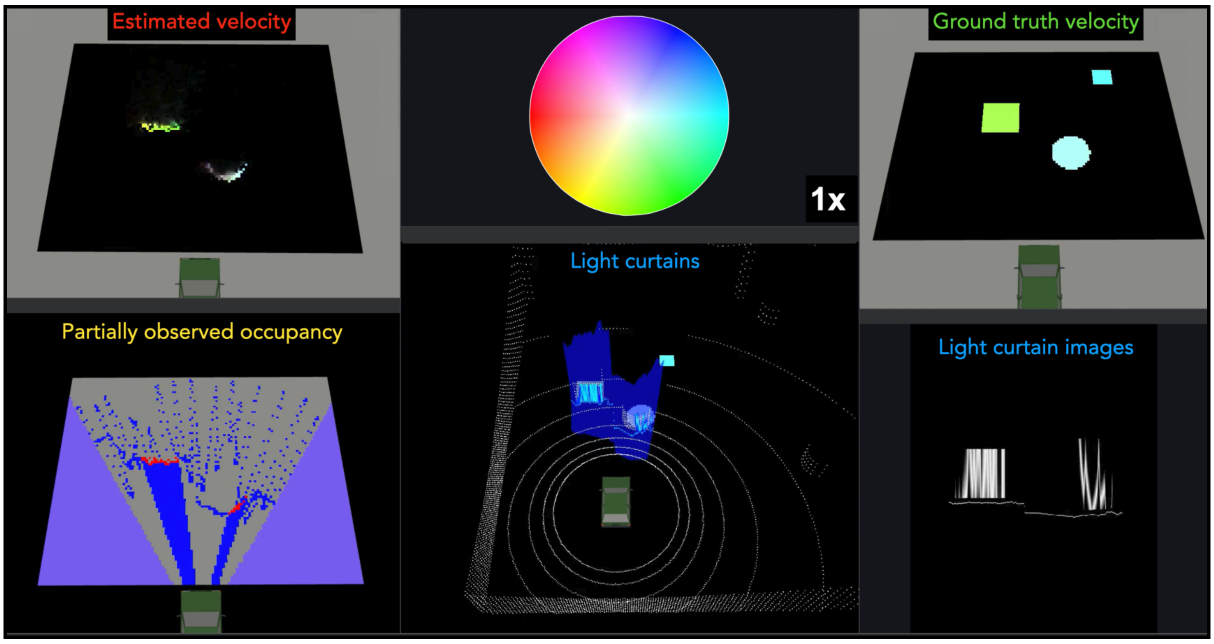
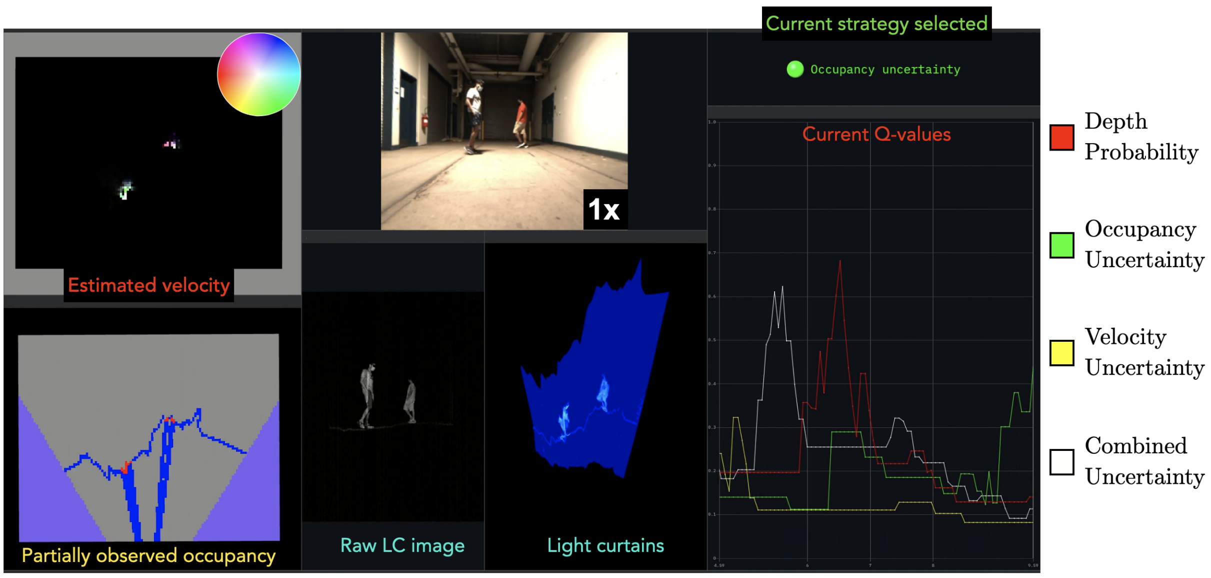
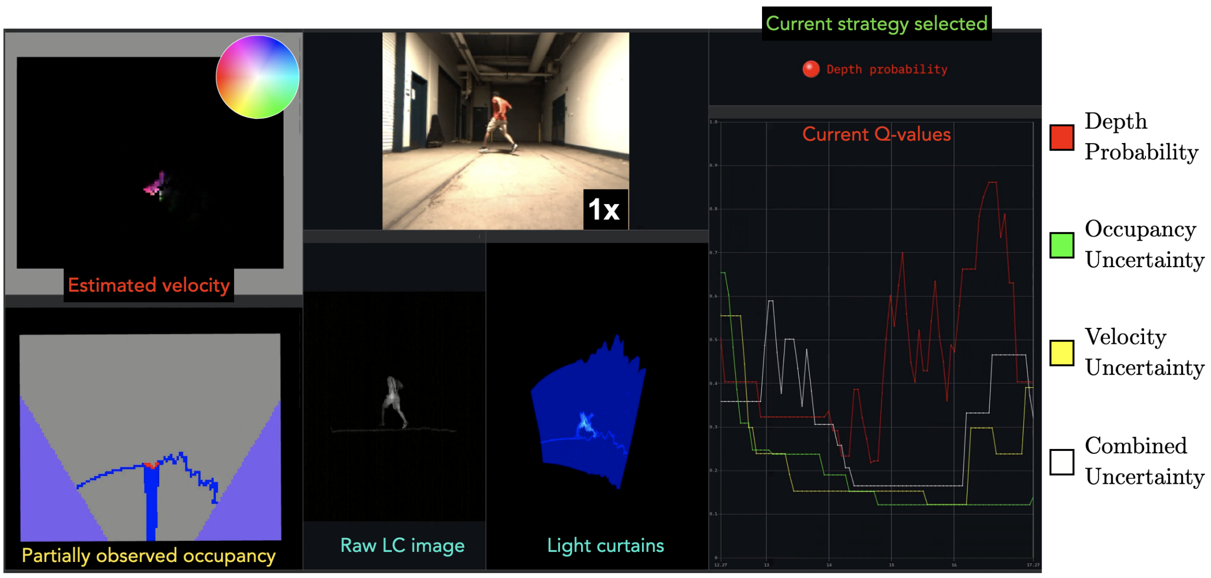


Figure 8.10: Velocity estimation in simulation. *Top-left:* estimated velocity and occupancy using the dynamic occupancy grid. *Top-right:* ground truth occupancy and velocity. *Top-center*: color-coding for visualizing velocity. *Bottom-left:* Partial occuapancy observations from light curtain measurements that are input to the dynamic occupancy grid. *Bottom-center:* intelligent curtains in blue. *Bottom-right:* raw light curtain images; intensities are high when objects surfaces intersect the curtain.



(a)



(b)

Figure 8.11: **Velocity estimation in the real world using multi-armed bandits (MAB)**. For both (a) and (b): *Top-right panel:* current strategy selected by the MAB. *Bottom-right:* the Q-values of each strategy. Higher Q-value is better; the action with the highest Q-value is chosen during exploitation. For an explanation of other panels, please refer to Fig. 8.10.
(a) Two pedestrians walking at a normal pace. The directions of motion are correctly inferred for each person as walking right (greenish-blue) and walking left (reddish). The current action selected maximizes occupancy uncertainty. (b) A harder environment where one person is running fast. The direction of velocity is correctly inferred as moving top-left (left and away from the sensor) i.e. reddish pink. The color saturation is high indicating the larger magnitude of velocity. The current action selected maximizes depth probability.

CHAPTER 9

Conclusion and Future Work

In this thesis, we have laid the algorithmic foundations of active perception using programmable light curtains. We investigated the use of light curtains for various perception tasks: object detection, depth estimation, obstacle detection and avoidance, and velocity estimation. We decomposed the overall architecture of our perception systems into three components: SENSE (sensing using light curtains potentially in conjunction with other sensors), THINK (state estimation and prediction from sensor measurements), and ACT (deciding where to place the light curtain). We proposed several methods for each component, namely, using sparse but accurate sensors (single-beam LiDARs), using dense but inaccurate sensors (cameras), deep learning based methods, Bayesian inference, Bayes filtering using dynamic occupancy grids (occupancy grids + particle filters), information gain, safety envelopes, multi-armed bandits with self-supervised rewards, and dynamic programming to incorporate light curtain constraints for computing curtains that maximize a task-specific objective, or to produce analytical detection probabilities for probabilistic safety guarantees.

In Chapter 4, we developed a method to incorporate the velocity and acceleration constraints of the light curtain device into the constraint graph. Then we developed a planning algorithm using dynamic programming that efficiently computes the feasible curtain that maximizes any task-specific objective while satisfying light curtian constraints. Furthermore, we showed how random curtains can be sampled from the constraint graph to discover obstacles with very high probability. We developed another dynamic programming based algorithm that analytically computes the probability of random curtains discovering obstacles, providing theoretical safety guarantees for obstacle detection and avoidance. The constraint graph and dynamic programming were used by all active perception tasks.

In Chapter 5, we looked at active object detection. We proposed using light curtain measurements in conjuction with a single-beam LiDAR to detect 3D bounding boxes around objects in a scene. We trained a deep-learning based 3D object detector to detect objects of the required semantic classes and output bounding boxes. We placed curtains at the locations of highest detector uncertainty to maximize information gain and sense more 3D points. This incrementally and consistently improved detection performance.

In Chapter 6, we performed depth estimation of a scene by combining measurements from a light curtain and a monocular RGB camera (a dense but inaccurate sensor) using a Bayesian inference framework. We used a monocular depth estimator to input an RGB image and output a depth distribution. We treated this distribution as a prior and combined it with likelihoods from light curtain measurements to refine the monocular depth estimates.

In Chapter 7, we estimated the safety envelope of the scene at each timetep. We trained a deep neural network to predict the location of the safety envelope in the next timestep from previous light curtain measurements alone. Then, we placed verification curtains at the predicted object locations to sense visible surfaces and estimate and track the safety envelope across time. At the same time, we continuously placed random curtains to discover obstacles.

In Chapter 8, we developed a method to explicitly predict the velocity of each point in the scene using light curtains. We used a probabilistic Bayes filtering approach using dynamic occupancy grids that combine conventional occupancy grids with particle filters. After estimating the velocity and occupancy of each location in the scene, we combined the information-gain based and verification-based placement strategies. We used a multi-armed bandit framework to intelligently switch between multiple strategies. This was enabled by a novel self-supervised reward function computed at test time using light curtain placements alone. We saw that this approach improved the accuracy of velocity and occupancy estimation over individual placement strategies used in isolation.

However, much more could be done to improve active perception using light curtains:

- Currently, dynamic occupancy grids have been implemented on a single-core CPU. Parallelizing the particle filter-based motion updates across multiple CPU cores or over a GPU could potentially improve the accuracy and efficiency of velocity estimation.

- In this thesis, we used an ϵ-greedy algorithm for multi-armed bandits to switch between multiple curtain placement strategies. It may be worth exploring other algorithms such as Thompson sampling and upper-confidence bound (UCB) that could potentially improve exploration.

- Another direction to improve multi-armed bandits is to incorporate context. Apart from allowing the rewards to be non-stationary, the current multi-armed bandit algorithm does not take any form of context into account. *Contextual bandits* take as input feature representations of the current context to make more informed decisions. Such representations could either be hand-designed or learned for selecting among multiple curtain placement strategies.

- The velocity estimates produced by dynamic occupancy grids are being used to compute where the next light curtain should be placed. However, these estimates could be used for other downstream tasks such as trajectory forecasting, obstacle avoidance and robot motion planning. They could be used to improve simultaneous localization and mapping (SLAM) systems which usually assume that all objects in the environment are stationary. Explicit velocity estimates could be first used to classify whether scene points are static or dynamic, and then to subtract dynamic objects from the scene. This could potentially improve the performance of SLAM systems.

- The intensity model of programmable light curtains can be improved by incorporating the geometry (surface normals) and reflectance properties of objects in the scene. Improving the light curtain sensor model and making it more physically accurate would likely improve the performance of Bayesian inference using light curtains.

- In this thesis, we treat light curtain placement as a short horizon problem, such as placing light curtains to maximize the information gain in the *immediate* next timestep, placing curtains to verify predicted object locations of the next timestep etc. Instead, we could model active perception using light curtains as a long-horizon partially observed Markov decision process (POMDP). Solving POMDPs is generally intractable even for very small state spaces. However, with advances in reinforcement learning algorithms and increased compute capacity, this research direction could potentially be promising for active light curtain perception.

With this thesis, we aim to lay the foundations and provide the initial set of tools for intelligently controlling programmable light curtains. However, we think these are only the first steps along a new and exciting research direction ripe with many interesting and unsolved problems for safe and accurate active robot perception using controllable sensors.

Bibliography

[APNH21] Siddharth Ancha, Gaurav Pathak, Srinivasa Narasimhan, and David Held. Active Safety Envelopes using Light Curtains with Probabilistic Guarantees. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[ARH$^+$20] Siddharth Ancha, Yaadhav Raaj, Peiyun Hu, Srinivasa G. Narasimhan, and David Held. Active perception using light curtains for autonomous driving. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 751–766, Cham, 2020. Springer International Publishing.

[AW$^+$87] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[Baj88] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

[BAT18] Ruzena Bajcsy, Yiannis Aloimonos, and John K Tsotsos. Revisiting active perception. *Autonomous Robots*, 42(2):177–196, 2018.

[BBB$^+$19] Andrea Bajcsy, Somil Bansal, Eli Bronstein, Varun Tolani, and Claire J Tomlin. An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1758–1765. IEEE, 2019.

[BLW20] A. W. Bergman, D. B. Lindell, and G. Wetzstein. Deep adaptive lidar: End-to-end optimization of sampling and depth completion at low sampling rates. In *2020 IEEE International Conference on Computational Photography (ICCP)*, pages 1–11, 2020.

[BMK14] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

[BWW$^+$19] Joseph R Bartels, Jian Wang, William Whittaker, Srinivasa G Narasimhan, et al. Agile depth sensing using triangulation light curtains. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7900–7908, 2019.

[BWWN19] Joseph R. Bartels, Jian Wang, William "Red" Whittaker, and Srinivasa G. Narasimhan. Agile depth sensing using triangulation light curtains. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[CAF18] Ricson Cheng, Arpit Agarwal, and Katerina Fragkiadaki. Reinforcement learning of active vision for manipulating objects under occlusions. In *Conference on Robot Learning*, pages 422–431. PMLR, 2018.

[CBL+19] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[CBL+20] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving, 2020.

[CC18] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[CKBP18] Arun CS Kumar, Suchendra M Bhandarkar, and Mukta Prasad. Depthnet: A recurrent neural network architecture for monocular depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 283–291, 2018.

[CLH+05] Howie Choset, Kevin M Lynch, Seth Hutchinson, George A Kantor, and Wolfram Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[CLS+19] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3d tracking and forecasting with rich maps, 2019.

[Con85] CI Connolly. The determination of next best views. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 432–435. IEEE, 1985.

[CZCZ21] Chao Cao, Hongbiao Zhu, Howie Choset, and Ji Zhang. Tare: A hierarchical framework for efficiently exploring complex 3d environments. In *Robotics: Science and Systems Conference (RSS), Virtual*, 2021.

[DB02] Joachim Denzler and Christopher M Brown. Information theoretic sensor data selection for active object recognition and state estimation. *IEEE Transactions on pattern analysis and machine intelligence*, 24(2):145–157, 2002.

[DC17] Jonathan Daudelin and Mark Campbell. An adaptable, probabilistic, next-best view algorithm for reconstruction of unknown 3-d objects. *IEEE Robotics and Automation Letters*, 2(3):1540–1547, 2017.

[DKMK16] Andreas Doumanoglou, Rigas Kouskouridas, Sotiris Malassiotis, and Tae-Kyun Kim. Recovering 6d object pose and predicting next-best-view in the crowd. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3583–3592, 2016.

[DON11] Radu Danescu, Florin Oniga, and Sergiu Nedevschi. Modeling and tracking the driving environment with a particle-based occupancy grid. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1331–1342, 2011.

[Elf89] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[EPF14] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[GAB17] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency, 2017.

[GFMP08] D. Gallup, J. Frahm, P. Mordohai, and M. Pollefeys. Variable baseline/resolution stereo. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[GJAB$^+$19] Tobias Gruber, Frank Julca-Aguilar, Mario Bijelic, Werner Ritter, Klaus Dietmayer, and Felix Heide. Gated2depth: Real-time dense lidar from gated images, 2019.

[GKOV17] Weihao Gao, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Estimating mutual information for discrete-continuous mixtures. *Advances in neural information processing systems*, 30, 2017.

[GLSU13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[GS15] Yoav Grauer and Ezri Sonn. Active gated imaging for automotive safety applications. In Robert P. Loce and Eli Saber, editors, *Video Surveillance and Transportation Imaging Applications 2015*, volume 9407, pages 112 – 129. International Society for Optics and Photonics, SPIE, 2015.

[GWCV16] A Gaidon, Q Wang, Y Cabon, and E Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016.

[HH12] Sebastian Haner and Anders Heyden. Covariance propagation and next best view planning for 3d reconstruction. In *European Conference on Computer Vision*, pages 545–556. Springer, 2012.

[Hub92] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, volume 76, pages 492–518. IEEE, 1992.

[HZHR20] Peiyun Hu, Jason Ziglar, David Held, and Deva Ramanan. What you see is what you get: Exploiting visibility for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11001–11009, 2020.

[ICG$^+$18] Eddy Ilg, Ozgun Cicek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. Uncertainty estimates and multi-hypotheses networks for optical flow, 2018.

[ISDS16] Stefan Isler, Reza Sabzevari, Jeffrey Delmerico, and Davide Scaramuzza. An information gain formulation for active volumetric 3d reconstruction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3477–3484. IEEE, 2016.

[KML$^+$18] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.

[KRBS15] Simon Kriegel, Christian Rink, Tim Bodenmüller, and Michael Suppa. Efficient next-best-scan planning for autonomous 3d surface reconstruction of unknown objects. *Journal of Real-Time Image Processing*, 10(4):611–631, 2015.

[LGK$^+$19] Chao Liu, Jinwei Gu, Kihwan Kim, Srinivasa G Narasimhan, and Jan Kautz. Neural rgb (r) d sensing: Depth and uncertainty from a video camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10986–10995, 2019.

[LVC$^+$19] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[MCCY18] Abdulla Mohamed, P. Culverhouse, A. Cangelosi, and C. Yang. Active stereo platform: online epipolar geometry update. *EURASIP Journal on Image and Video Processing*, 2018:1–16, 2018.

[MHG$^+$14] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[MLK$^+$19] Gregory P Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12677–12686, 2019.

[MSK] Larry Matthies, Richard Szeliski, and Takeo Kanade. Depth maps from image sequences1.

[MST$^+$20] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[NLMW20] M. Nishimura, David B. Lindell, Christopher A. Metzler, and G. Wetzstein. Disambiguating monocular depth estimation with a single transient. In *ECCV*, 2020.

[NMH$^+$05] Y. Nakabo, Toshiharu Mukai, Yusuke Hattori, Y. Takeuchi, and N. Ohnishi. Variable baseline stereo tracking vision system using high-speed linear slider. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1567–1572, 2005.

[PTF$^+$20] Francesco Pittaluga, Zaid Tasneem, Justin Folden, Brevin Tilmon, Ayan Chakrabarti, and Sanjeev Koppal. Towards a mems-based adaptive lidar, 10 2020.

[PVGDVG20] Vaishakh Patil, Wouter Van Gansbeke, Dengxin Dai, and Luc Van Gool. Don't forget the past: Recurrent depth estimation from monocular video. *IEEE Robotics and Automation Letters*, 5(4):6813–6820, 2020.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[RR17] Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. 2017.

[RVBR18] Charles Richter, William Vega-Brown, and Nicholas Roy. Bayesian learning for safe high-speed navigation in unknown environments. In *Robotics Research*, pages 325–341. Springer, 2018.

[RWR14] Charles Richter, John Ware, and Nicholas Roy. High-speed autonomous navigation of unknown environments using learned probabilities of collision. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6114–6121. IEEE, 2014.

[SAMR18] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SMAG18] Martin Simony, Stefan Milzy, Karl Amendey, and Horst-Michael Gross. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[SRR03] William R Scott, Gerhard Roth, and Jean-François Rivest. View planning for automated three-dimensional object reconstruction and inspection. *ACM Computing Surveys (CSUR)*, 35(1):64–96, 2003.

[SST18] A. Schneider, N. Sharma, and Bryan Tripp. Visually guided vergence in a new stereo camera system. 2018.

[SWL19] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[Thr02] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[TJFK20] B. Tilmon, E. Jain, S. Ferrari, and S. Koppal. Foveacam: A mems mirror-enabled foveating camera. In *2020 IEEE International Conference on Computational Photography (ICCP)*, pages 1–11, 2020.

[TWXS18] Zaid Tasneem, Dingkang Wang, Huikai Xie, and Koppal Sanjeev. Directionally controlled time-of-flight ranging for mobile sensing platforms. 06 2018.

[TZEM17] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2626–2634, 2017.

[VGSMCLD14] J Irving Vasquez-Gomez, L Enrique Sucar, Rafael Murrieta-Cid, and Efrain Lopez-Damian. Volumetric next-best-view planning for 3d object reconstruction with positioning error. *International Journal of Advanced Robotic Systems*, 11(10):159, 2014.

[WBW$^+$18] Jian Wang, Joseph Bartels, William Whittaker, Aswin C Sankaranarayanan, and Srinivasa G Narasimhan. Programmable triangulation light curtains. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[WGRD20] Stefanie Walz, Tobias Gruber, Werner Ritter, and Klaus Dietmayer. Uncertainty depth estimation with gated images for 3d reconstruction, 2020.

[Wil94] David Wilkes. *Active object recognition*. 1994.

[WPF19] Rui Wang, Stephen M Pizer, and Jan-Michael Frahm. Recurrent neural network for (un-) supervised learning of monocular video visual odometry and depth. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5555–5564, 2019.

[WSK$^+$15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[XSC19] Zhihao Xia, Patrick Sullivan, and Ayan Chakrabarti. Generating and exploiting probabilistic monocular depth estimates, 2019.

[YHR19] Gengshan Yang, Peiyun Hu, and Deva Ramanan. Inferring distributions over depth from a single image. In *Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems*, November 2019.

[YKI$^+$18] T. Yamamoto, Y. Kawanishi, I. Ide, H. Murase, F. Shinmura, and D. Deguchi. Efficient pedestrian scanning by active scan lidar. In *2018 International Workshop on Advanced Image Technology (IWAIT)*, pages 1–4, 2018.

[YLL$^+$18] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo, 2018.

[YLU18] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *Conference on Robot Learning*, pages 146–155, 2018.

[YML18] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018.

[ZBGGV$^+$19] Javad Zolfaghari Bengar, Abel Gonzalez-Garcia, Gabriel Villalonga, Bogdan Raducanu, Hamed H Aghdam, Mikhail Mozerov, Antonio M Lopez, and Joost van de Weijer. Temporal coherence for active learning in videos. *arXiv preprint arXiv:1908.11757*, 2019.

[ZCX$^+$21] Hongbiao Zhu, Chao Cao, Yukun Xia, Sebastian Scherer, Ji Zhang, and Weidong Wang. Dsvp: Dual-stage viewpoint planner for rapid exploration by dynamic expansion. In *International Conference on Intelligent Robots and Systems (IROS), Virtual*, 2021.

[ZGW$^+$18] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 340–349, 2018.

[ZJZ$^+$19] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019.

[ZSL$^+$19] Haokui Zhang, Chunhua Shen, Ying Li, Yuanzhouhan Cao, Yu Liu, and Youliang Yan. Exploiting temporal consistency for real-time video depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1725–1734, 2019.

[ZSZ$^+$20] Chao Qiang Zhao, Qi Yu Sun, Chong Zhen Zhang, Yang Tang, and Feng Qian. Monocular depth estimation based on deep learning: An overview. *Science China Technological Sciences*, pages 1–16, 2020.

[ZT18] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.