Programmable light curtains for Safety Envelopes, SLAM and Navigation

Gaurav Pathak

CMU-RI-TR-22-56

August 17, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Prof. Srinivas Narasimhan, *co-chair*
Prof. David Held, *co-chair*
Prof. Ji Zhang
Venkata Siddharth Ancha

Abstract

Conventional robot perception and navigation pipelines are built using traditional sensors such as RGB cameras, stereo depth sensors and LiDARs. These sensors scan the entire scene in a fixed and uniform way. In contrast, programmable light curtains are a recently-invented, resource-efficient sensor that measure the depth of any vertically-ruled surface ("curtain") specified by the user. Compared to LiDARs, light curtains are relatively inexpensive, significantly faster (45-60 Hz) and capture depth at a much higher resolution (640 scan lines). However, they require user control.

The main contributions of this thesis are to (1) integrate programmable light curtains with an existing, state-of-the-art navigation and autonomy stack, (2) develop algorithms for enabling light curtains to detect and avoid obstacles for safe navigation, and (3) perform high resolution mapping and accurate robot localization using intelligent curtain placements. Our overall system consists of parallelized components that interact naturally and continuously while running at their own independent speeds. This work is a step towards full-stack autonomous robot navigation using fast, high-resolution, controllable sensing. We demonstrate our integration on a wheelchair robot.

Acknowledgments

First, I would like to thank my advisors Prof. Srinivas Narasimhan and Prof. David Held for all their help, guidance, and support during my masters. Their valuable insights were crucial in shaping every aspect of this thesis. I would like to extend my sincere gratitude to Siddharth Ancha. His guidance, camaraderie and long discussions about problems were essential during the course of my masters. Next, I would like to thank my committee member Prof. Ji Zhang for many helpful discussions about SLAM and Path Planning.

I would like to thank all my lab members for many fruitful discussions. I would be remiss if I do not mention the help of countless friends. They were always there when I needed them. Finally, I would like to thank my parents and family for all their support and encouragement.

The project Active Safety Envelopes using Light Curtains with Probabilistic Guarantees was done in collaboration with Siddharth Ancha. This work was supported by the National Science Foundation under Grants No. IIS-1849154, IIS-1900821 and by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092. The second project is funded in part by Carnegie Mellon University's Mobility21 National University Transportation Center, which is sponsored by the US Department of Transportation.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

LiDARs and depth cameras are the most commonly used 3D sensors. But they suffer from various drawbacks. LiDARs have low resolution. Further, 3D LiDARs are typically very expensive and they only operate at low frequencies of about 5-20Hz. Depth cameras have their own set of problems: they only operate accurately at low ranges, typically only work reliably in indoor settings, are sensitive to ambient lighting, fast motions and struggle in low texture environments [23]. In contrast, Programmable light curtains are a recently invented controllable 3D sensor that combines the advantages of a LiDAR and depth camera [5]. Unlike LiDARs, Programmable light curtains have high resolution and operate at a frequency of about 45 to 60 Hz. They are also much cheaper. In comparison to depth cameras, Programmble light curtains work reliably indoors and more importantly outdoors owing to the ambient light subtraction module and the use of line laser.

Another key difference between Programmable light curtains and conventional 3D sensors is the sensing paradigm. Conventional 3D sensors sense the scene globally and passively. The entire scene is sensed equally in a fixed pattern and at all times. On the other hand, light curtains are active and local sensors. The sensor outputs dense depth at user specified locations. While we can obtain high resolution depth measurements from light curtains, it requires algorithmic work from the user to intelligently place these curtains based on the task.

The primary contribution of this thesis are to track the motion of moving obstacles using light curtains and to integrate Programmable light curtains in the autonomy

stack. To that end, in the first chapter, we address the problem of tracking motion of dynamic obstacles. We tackle this using light curtains and estimate the "Safety Envelopes" of the scene. Safety Envelopes are defined as imaginary, vertically ruled surfaces that demarcate the boundary between safe and unsafe regions for robot navigation. We compute these safety envelopes using current estimates of objects location obtained by placing random curtains in the scene and forecasting the estimates of the previous timesteps. We perform extensive qualitative and quantitative experiments to demonstrate the effectiveness of our method. In the second chapter, we describe the methodology to integrate light curtains with ORB-SLAM [6] and AEDE [7]. ORB-SLAM is one of the state of the art SLAM system which supports monocular, stereo, RGB-D and visual inertial SLAM. We integrate with RGB-D module where the depth is obtained using Programmabale light curtains. We then use state estimation from this integration and detections from light curtains to perform path-planning and low level control using AEDE [7]. This pipeline allows us to navigate safely in static environments.

Chapter 2

Active Safety Envelopes using Light Curtains with Probabilistic Guarantees

2.1 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?

Given a LiDAR sensor, the locations of obstacles can be computed from the captured point cloud; however, LiDARs are typically expensive and low-resolution. 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 and does not guarantee safety.

An alternative approach is to use *active perception* [3, 4], where only the important

and required parts of the scene are accurately sensed, by actively guiding a controllable sensor in an intelligent manner. Specifically, a programmable light curtain [1, 5, 43] is a light-weight *controllable* sensor that detects objects intersecting any user-specified 2D vertically ruled surface (or a 'curtain'). 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).

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 [5] 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 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 systematic framework to generate random curtains that respect the physical constraints of the light curtain device, by extending the "light curtain constraint graph" introduced in prior work [1] (Sec. 2.4, 2.5.1).
2. 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. 2.5.2, 2.7.1).
3. We combine random light curtains with a machine learning based forecasting model to efficiently estimate safety envelopes (Sec. 2.6).
4. 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. 2.7.2).

2.2 Related Work

2.2.1 Active perception and light curtains

Active perception involves actively controlling a sensor for improved perception [3, 4], such as controlling camera parameters [3], moving a camera to look around occlusions [9], and next-best view planning [10]. The latter refers to approaches that select the best sensing action for specific tasks such as object instance classification [14, 15, 40, 45] and 3D reconstruction [13, 22, 24, 42]. Light curtains were introduced in prior work [5, 43] as an adaptive depth sensor. Prior work has also explored the use of light curtains. Ancha et al. [1] introduced the light curtain constraint graph

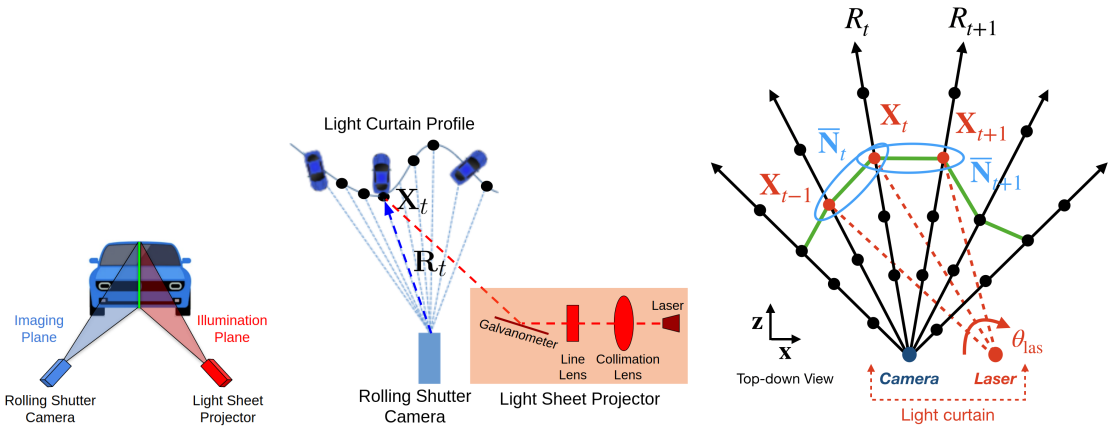
to compute feasible light curtains. Bartels et al. [5] were the first to empirically use random curtains to quickly discover objects in a scene. However, there are several key differences from our work. First, we solve a very different problem: while Ancha et al. [1] use light curtains to perform active 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 [1, 5] 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.

2.2.2 Multi-frame depth estimation

There is a large body of prior work on depth estimation across multiple frames [12, 26, 27, 29, 44, 46, 47]. Liu et al. [26] aggregate per-frame depth estimates across frames using Bayesian filtering. Matthies et al. [27] use a similar Bayesian approach, but their method is only applied to controlled scenes and restricted camera motion. Other works [12, 29, 44, 47] 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. To the best of our knowledge, we are the first to use a controllable sensor to directly estimate the safety envelope of the scene.

2.2.3 Safe navigation

Many approaches for safety guaranteed navigation use 3D sensors like LiDARs [33, 34, 39] and/or cameras [2, 32]. The sensor data is converted to occupancy grids/maps [2, 33, 34]; safety and collision avoidance guarantees are provided for planning under these representations. Other works use machine learning models to recognize unsafe,



(a) Working principle (b) Optical schematic (top view) (c) Extended constraint Graph

Figure 2.1: (a, b) Illustration of programmable light curtains adapted from [1]. (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. (c) 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.

out-of-distribution inputs [32] or learning to predict collision probabilites [33, 34]. 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.

2.3 Background on light curtains

Programmable *light curtains* [1, 5, 43] are a recently developed sensor for controllable depth sensing. "Light curtains" can be thought of as virtual surfaces placed in the environment that detect points on objects intersecting this surface. The working principle is illustrated in Fig. 2.1(a, b). The device sweeps a vertically ruled surface by rotating a light sheet laser using a galvo-mirror synchronously with the vertically aligned camera's rolling shutter. Object points intersecting a vertical line of the ruled surface are imaged brightly in the corresponding camera column. We denote the top-down projection of the imaging plane corresponding to the t-th pixel column as a

"camera ray" R_t. The rolling shutter camera successively activates each image plane (column), corresponding to rays $R_1, \ldots, R_T$ from left to right, with a time difference of Δt between successive activations. The top-down projection of the vertical line intersecting the t-th imaging plane lies on R_t and will be referred to as a "control point" $\mathbf{X}_t$.

Input: A light curtain is uniquely defined by where it intersects each camera ray R_t in the top-down view, i.e. the set of control points $(\mathbf{X}_1, \cdots, \mathbf{X}_T)$, one for each camera ray. This is the input to the light curtain device. Then, to image $\mathbf{X}_t$ on camera ray R_t, the galvo-mirror is programmed to rotate by an angle of $\theta(\mathbf{X}_t)$ that is required for the laser sheet to intersect R_t at $\mathbf{X}_t$. By specifying a control point $\mathbf{X}_t$ for each camera ray, the light curtain device can be made to image any vertically ruled surface [5, 43].

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

2.4 Generating Feasible Light Curtains

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 $\mathbf{\Omega}_t = (\theta(\mathbf{X}_t) - \theta(\mathbf{X}_{t-1}))/\Delta t$ 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:

$$|\theta(\mathbf{X}_t) - \theta(\mathbf{X}_{t-1})| \leq \omega_{\max} \cdot \Delta t \qquad 2 \leq t \leq T \tag{2.1}$$

$$|\theta(\mathbf{X}_{t+1}) + \theta(\mathbf{X}_{t-1}) - 2\theta(\mathbf{X}_t)| \leq \alpha_{\max} \cdot (\Delta t)^2 \qquad 2 \leq t < T \tag{2.2}$$

In order to compute feasible light curtains that satisfy physical constraints, Ancha et al. [1] introduced a "light curtain constraint graph," denoted 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. (2.1)).

Ancha et al. [1] 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. 2.2, 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. 2.1 (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 (2.1, 2.2). 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.

2.5 Random curtains & theoretical guarantees

Recall that the light curtain will only sense the parts of the scene where the curtain is placed. Thus we must decide where to place the curtain in order to sense the scene and estimate the safety envelope. Our proposed method uses a combination of random curtains as well as learned forecasting to estimate the safety envelope of an unknown scene. In this section, we show how a random curtain can be sampled from the extended constraint graph $\overline{\mathcal{G}}$ and how to analytically compute the probability of a random curtain detecting an obstacle, which helps to probabilistically guarantee obstacle detection of our overall method.

2.5.1 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 A.0.1. Our random curtain sampling process provides the flexibility to design any initial and transition probability distribution. See Appendix A.0.1 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.

2.5.2 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. 2.5.1) 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

$$P(\mathbf{D}) = \sum_{\mathbf{X}_1, \mathbf{X}_2} P_{\text{det}}(\mathbf{X}_1, \mathbf{X}_2)\ P(\mathbf{X}_1, \mathbf{X}_2). \tag{2.3}$$

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:

$$P_{\text{det}}(\mathbf{X}_{t-1}, \mathbf{X}_t) = \begin{cases} 1 & \text{if } \mathbf{D}_t(\mathbf{X}_t, O_t) = 1 \\ \sum_{\mathbf{X}_{t+1}} P_{\text{det}}(\mathbf{X}_t, \mathbf{X}_{t+1})\ P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t) & \text{otherwise} \end{cases} \tag{2.4}$$

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. 2.4). Finally, after obtaining the probabilities for nodes on the second ray, the overall curtain detection probability can be computed as described using Eqn. 2.3. Pseudocode for this method can be found in Algorithm 2 in Appendix A.0.2. 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 A.0.3.

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 2.7.1, 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.

2.6 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 A.0.5 for details of our efficient, parallelized implementation of random curtain placement and forecasting, as well as an analysis of the pipeline's runtime.

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 A.0.4.

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 [36], a widely-used imitation learning algorithm to train a policy with expert or ground-truth supervision across multiple timesteps. We use the Huber loss [21] 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.

2.7 Experiments

2.7.1 Random curtain analysis

In this section, we use the dynamic programming approach introduced in Section 2.5.2 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 2.2, 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 2.3 (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 2.3 (b) shows the multi-curtain detection probabilities for objects from the KITTI [19] 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.

2.7.2 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 [53], a large, simulated dataset containing photorealistic scenes of urban driving scenarios. It

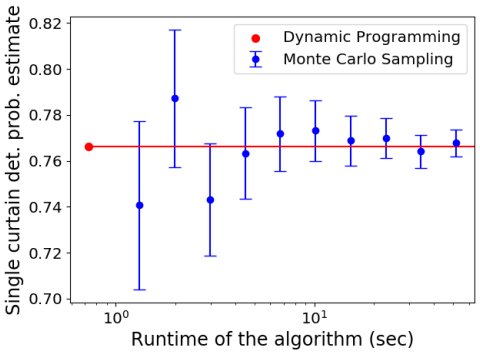


Figure 2.2: Comparison of our dynamic programming approach with a Monte Carlo sampling based approach to estimate the probability of a random curtain detecting an object of size $2m \times 2m$. The x-axis shows the runtime of the methods (in seconds), and the y-axis shows estimated detection probability. 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.

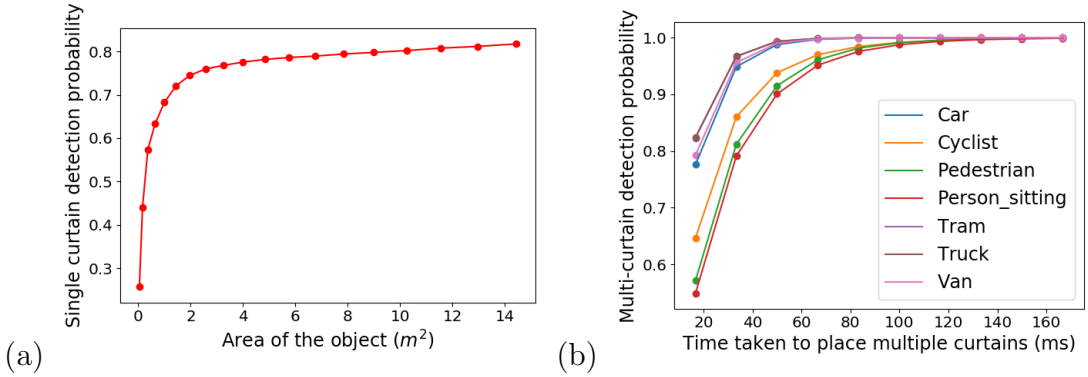


Figure 2.3: (a) The probability of random curtains detecting objects of various areas. For an object of a fixed area, we average the probability across various orientations of the object. Larger objects are detected with higher probability. (b) We show the detection probability of canonical objects from classes in the KITTI [19] 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.

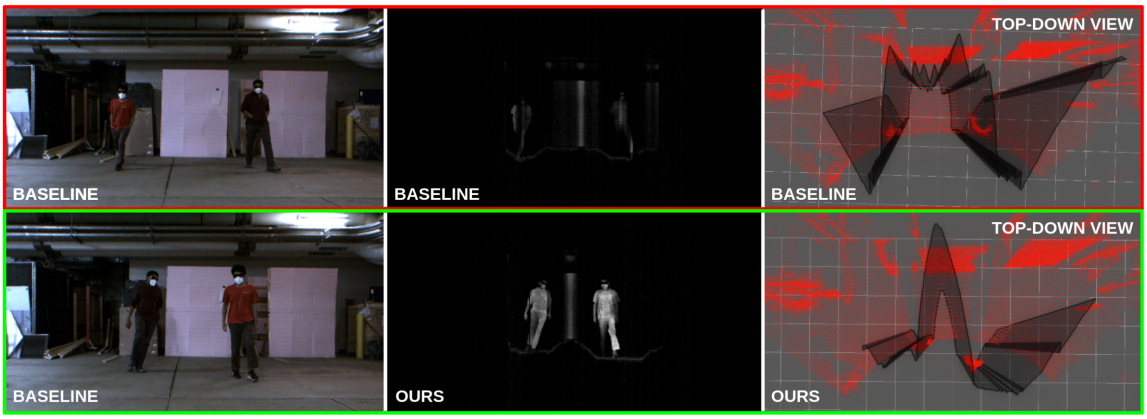


Figure 2.4: Qualitative results in a real-world environment with two walking pedestrians, comparing a hand-crafted baseline policy (top-row) with our method (bottom-row). *Left column:* contains RGB scene images. *Middle column:* contains the light curtain images, where higher intensity means a closer 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.

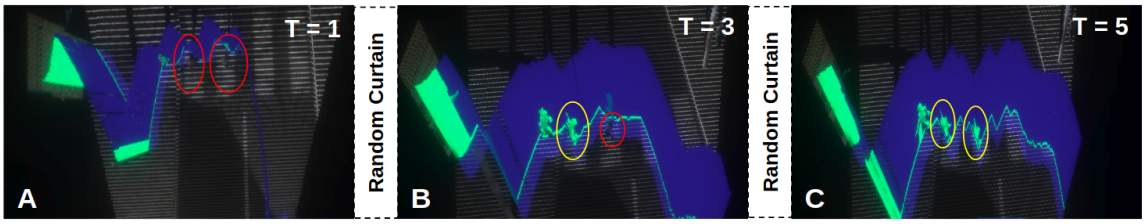


Figure 2.5: We illustrate the benefits of placing random curtains (that come with probabilistic guarantees of obstacle detection) while estimating safety envelopes, shown in SYNTHIA [53], a simulated urban driving environment. The blue surfaces are the estimated safety envelopes, and the green points 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.

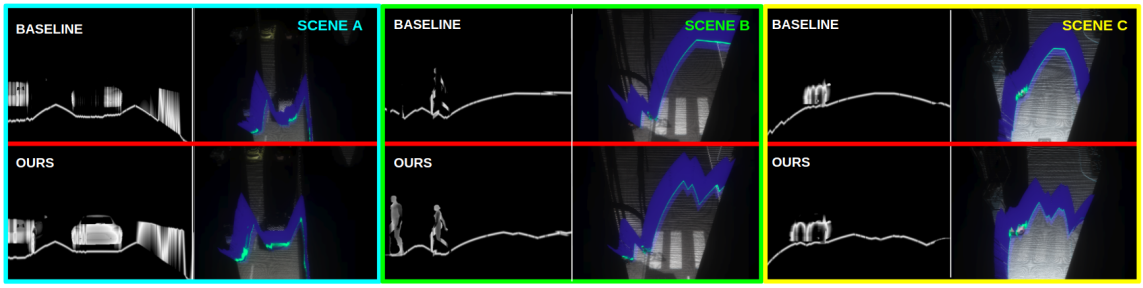


Figure 2.6: Comparison of the safety envelope estimation between a hand-crafted baseline policy (top row) and a trained neural network (bottom row), in three simulated urban driving scenes from the SYNTHIA [53] dataset. For each scene and method, the left column shows the intensity image of the light 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.

	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)
	↓	↓	↓	↓	↓	↓	↑	↑	↑
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	0.0896	1.7124	0.1372	0.0731	0.1101	0.7076	0.7159	0.7900	0.8138
1D-GNN	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 2.1: Performance of safety envelope estimation on the SYNTHIA [53] 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)
		↓	↓	↓	↓	↓	↓	↑	↑	↑
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 2.2: 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.

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 [16, 52]. The metrics are averaged over multiple timesteps to evaluate the policy's performance across time.

Baselines: In Table 2.1, we compare our method to the hand-crafted policy described in Sec. 2.6. 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. Please see Appendix A.0.4 for more details about their network architectures. See Table 2.1 for a comparison of our method with the baselines in the SYNTHIA environment, and Table 2.2 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. 2.1, "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 A.0.6 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. 2.1, "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. 2.1, "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. 2.4, and in the SYNTHIA [53] simulated environment in Figs. 2.5, 2.6. 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.

2.8 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 3

Programmable light curtains for Simultaneous Localization, Mapping, and Navigation

3.1 Motivation

We aim to develop an autonomous navigation robot using the relatively low cost sensors "Programmable Light Curtains" 3.1. We first look at what are the components required for autonomy. To that end, one of the most fundamental component would be a "Simultaneous Localization and Mapping" system. A robot should know where it is (localization) in its environment (mapping) to estimate velocity of other objects, avoid obstacles, safely plan its path and reach a goal destination. To equip our robot with a SLAM system we integrated light curtains [5] with ORB-SLAM [6].

3.2 Background on Visual SLAM

Visual SLAM uses a camera to estimate the pose of the robot and the environment map by tracking the changes in structure of the environment induced by the motion of the robot. Visual SLAM methods are gaining popularity due to relatively cheap cost of the cameras as opposed to a LIDAR. There have been works which alleviate

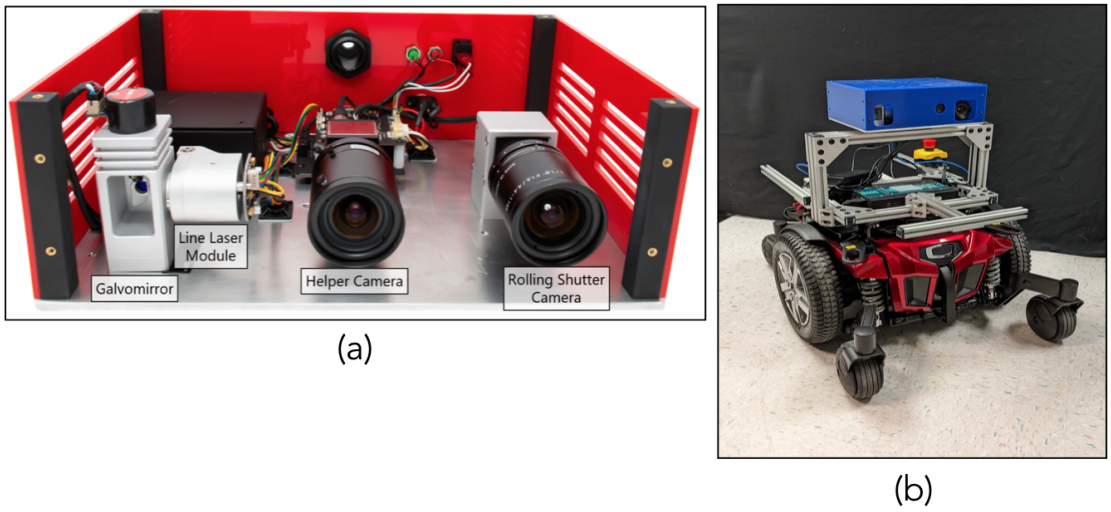


Figure 3.1: (a) Programmable Light curtains [5], (b) A remote controlled wheelchair outfitted with Programmable light curtains.

the problem of using expensive LIDARs by making a custom 3D LIDAR using a 2-axis one [48] but this requires significant engineering effort. Further, by fusing the data from other sensors such as IMUs and depth cameras with the on-board RGB camera, the accuracy of Visual SLAM is approaching to that of the LIDARs.

There are two main approaches to Visual SLAM that exists in literature: Direct visual SLAM [28], [17], [11] and Indirect visual SLAM [6], [31], [30]. The major difference between these two methods stems from the way they handle the input images. Indirect methods abstract out the input images by first detecting features such as corners and then completely discarding the images. The motion of the robot and sparse map of the environment is generated using re-projection errors. Whereas, direct methods operate on the entire images without using any abstraction. They directly minimize the photometric errors (intensity difference) and generate a semi-dense map of the environment by using filtering of stereo pairs. The speed of operation of indirect methods is usually faster than that of the direct methods because of using feature abstraction and since we wanted our system to operate in real time we choose to use a state of the art Indirect method: ORB-SLAM [6].

One of the major problem of Visual SLAM systems estimating the motion of the robot and environment map just using on-board monocular camera is that of scale

ambiguity. In the literature, this problem is resolved by using depth sensors such as LIDARs, RGBD cameras [51], [49] and Inertial Measurement Unit (IMU) [18] [38]. RGB-D cameras though can resolve scale ambiguity are limited in their range of operation, have low precision and struggle in low texture and over-saturated environments. Whereas, IMU readings are not reliable when the robot is moving at constant velocity. Further, IMU sensors have high bias which needs to be estimated periodically adding extra computation cost albeit small [18]. To solve the problem of scale ambiguity in our system, we use a novel sensor: Programmable Light Curtains [5], 2.3. Programmable Light curtains [5] are a controllable depth sensor that work reliably indoors and outdoors but comes with additional challenge of providing the area of interest to sense and compute depth. We discuss various techniques to specify this area of interest in the Methods section 3.4.

3.3 Background on ORB-SLAM

ORB-SLAM [6] is one of the state of the art visual SLAM system. It supports visual and visual-inertial localization and mapping with Monocular, RGB-D, and stereo cameras. Further, it also supports relocalization and multi-map reuse. The system is based on the framework of Pose-Graph optimization [20], [25] and uses ORB-features [37] throughout its pipeline for tasks such as correspondence establishment, place-recognition, loop detection etc.

In this section, we give overview of the ORB-SLAM system (for full details please refer to [6]). The basic pipeline of the system consists of three threads running in parallel: Motion Estimation, Local Mapping and Loop Closure. Motion Estimation thread is responsible for predicting and refining the pose of the camera. Local Mapping thread is responsible for generating the outlier-free sparse map of the environment. Loop closure thread continuously works on detecting loops in the environment and performs a full bundle adjustment to refine poses and the map if necessary. One of the highlights of the system is, all the three threads use different length sliding window bundle adjustment [41] method for estimating the desired quantities. Further, to efficiently get the keyframes used in pose-graph optimization and bundle adjustment it maintains an efficient minimum-spanning tree data-structure called "essential graph". This provides efficient lookup and real time operation.

3.4 Method

In this section, we first describe the challenge involved in initialization of the ORB-SLAM system using a local depth sensor (Programmable Light Curtains [5]) and how do we resolve it. Next, we outline various techniques we used to place the light curtains for successive frames leveraging the insights gained from understanding of how ORB-SLAM uses depth information.

3.4.1 Initialization using Planar Sweeps

ORB-SLAM [6] assumes a global sensing paradigm. The initial map of the environment is directly initialized with the first frame received from a RGB-D camera. Then for each successive frames, the camera pose is computed by minimizing the reprojection error, between the detected ORB features and the corresponding map points, in a bundle adjustment framework. To provide a robust estimate of the pose, there should be sufficient number of correspondences. This implicitly requires the initial map to be dense. To this end, to successfully integrate a local depth sensor – Programmable Light curtains – into the pipeline we perform volumetric planar sweeps. Specifically, we design a planar curtain and place it at small increments of depths between a minimum and a maximum depth range. Let the set of all the accumulated curtains after this procedure be $\{(P_i, Q_i)\}_{i=1}^{N}$ and, where each P_i is a 3-channel and each Q_i is a 1-channel $H \times W$ image. Each pixel P_i contains the 3D location of the detected point and each pixel in Q_i is the corresponding intensity. From this, we generate the dense map by 3.1. The resulting dense map is visualized in Figure 3.2

$$d(u, v) = P_{f(Q(u,v))}(u, v) \quad u \in \{1, \cdots, W\}, v \in \{1, \cdots, H\} \tag{3.1}$$

$$f(Q(u, v)) = \text{argmax}_{i \in (1,N)} Q_i(u, v) \tag{3.2}$$

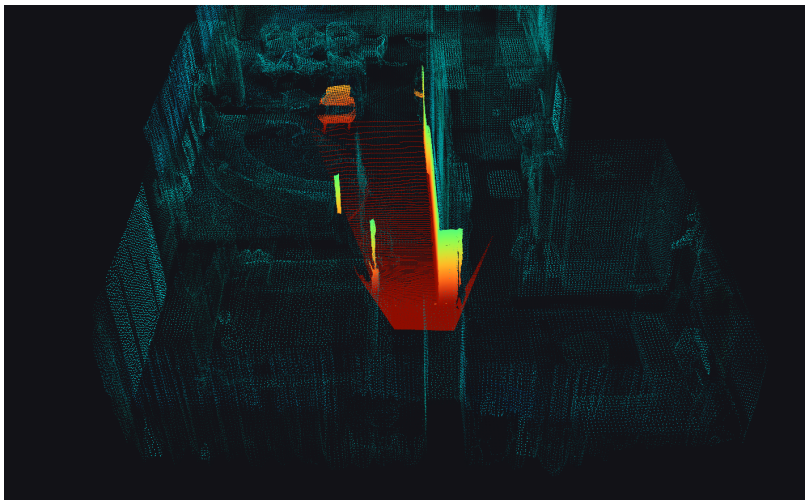


Figure 3.2: Result of dense map reconstruction using volumetric planar sweep. The colored pointcloud is the reconstructed dense map using planar sweep. It is overlayed on the ground truth pointcloud of a matterport house from [7]

3.4.2 Different Curtain Policies explored for providing depth in each frame.

Fixed depth planar curtains policy: This is one of the simplest placement strategies. Here, we just use a planar curtain which senses an area at a fixed distance from the current position of the robot. More formally, for each of the control point 2.3 we just specify a constant range to sense.

Random Curtains: The fixed depth planar curtains senses quite a small part of the scene at any given time. So to improve the coverage of the scene, another strategy that can be used is placing random curtains. Since, light curtains have velocity and acceleration constraints, we can only specify ranges along adjacent control points which satisfy these constraints. The procedure to generate such feasible random curtains is described in 2.5.1. Further, to avoid traversing the constraint graph each time we wish to generate a random curtain we pre-compute a lot of feasible random curtains offline.

Planar Sweeps poilcy: Even though random curtains provide good coverage of the scene and it is very hard for the object to avoid getting detected by it, they miss a lot of fine grained details of the objects shape. So to provide good coverage

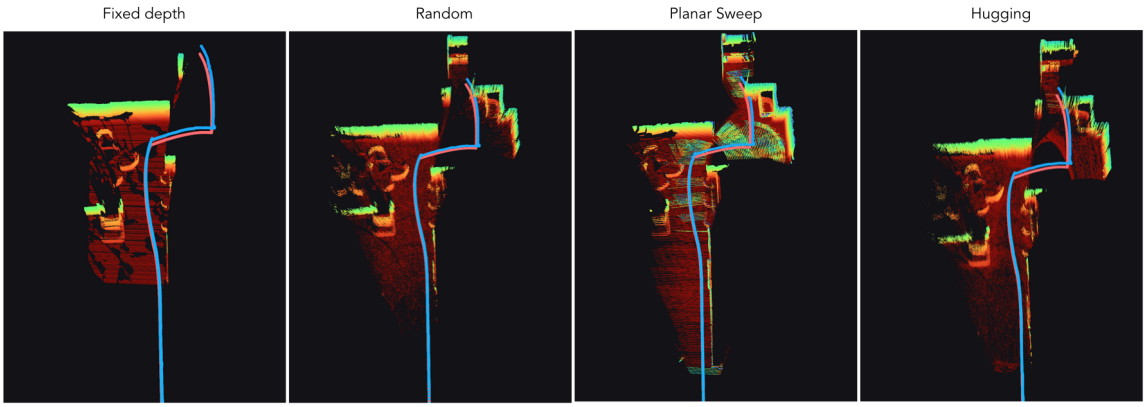


Figure 3.3: Results of ORB-SLAM with all the curtain placement policies are visualized in matterport house from [7]. Ground truth trajectory (Pink) and estimated trajectory (Blue) are overlayed on the reconstructed map. From figure we observe that the map reconstructed map from hugging policy is more dense and detailed as compared to others. We also observe that since the commanded ground truth trajectory does not contain a lot of large rotations, the performance of all the policies in estimating the pose of the robot is similar.

and also preserve find grained details for reconstruction of the environment map we adopt planar sweeps policy. As opposed to fixed depth planar curtains policy, here we continuously sweep a planar curtain within a volume specified with respect to the current position of the robot.

Hugging policy: In this policy, our goal is to place a curtain which intersects with all the objects in the scene. With the correct placement of such a curtain we would get a dense depth map. For this, we utilize particle filter based probabilistic dynamic occupancy grids. Essentially, we take the current state of the dynamic occupancy grids and perform ray casting to compute the estimated range for each of the control point.

3.5 Experiments and Results

Simulation Environment: We adapt AEDE [7] for qualitatively and quantitatively assessing our Programmable Light Curtains based Simultaneous localization and mapping system. We add the capability to simulate the output of the light curtains to AEDE using a simulated depth camera.

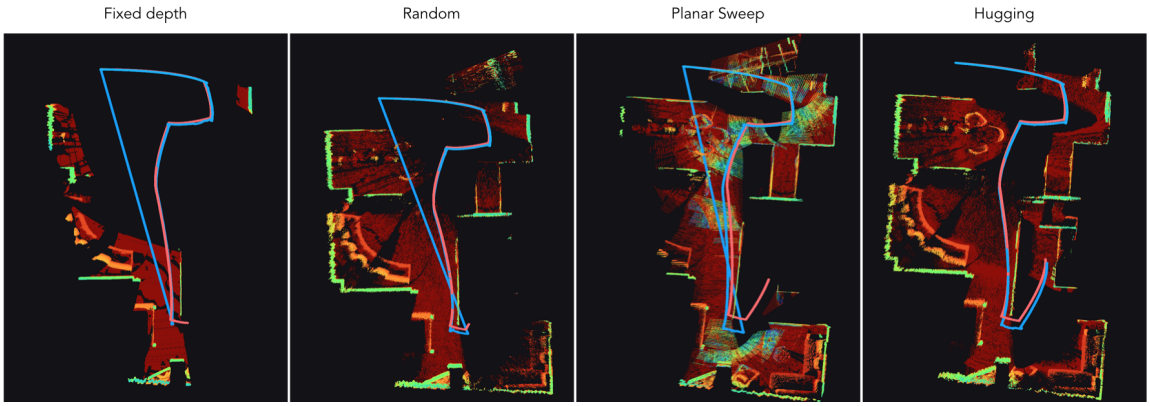


Figure 3.4: The results of all the curtain placement policies on a different trajectory are visualized. The commanded trajectory contains more number of large rotations as compared to trajectory in 3.3. As we see from the figure, only hugging policy successfully estimates the pose of the robot.

We evaluate all the curtain placement strategies described in 3.4 on a matterport house environment on two trajectories (Matterport 1 and Matterport 2) and an outdoor forest environment. For quantitative evaluation, we report Absolute Trajectory Error (ATE) over the entire trajectory as is standard in the SLAM literature [6] for Matterport 1 and Forest environment. For Matterport 2, since most of the policies fail to finish we report the number of successes of each policy over 5 runs. The qualitative results are shown in Figure 3.3 and Figure 3.4. As we can see from the figures, the hugging policy generates the most dense map of the environment. Further, we also observe that it is the only policy which succeeds in faithfully estimating the pose of the robot when the required path undergoes large in plane rotations 3.4, 3.2. This indicates that providing more depth is useful in environments where the robot has to undergo large rotations to reach the goal point. We also note that, for paths that do not require the robot to undergo large in plane rotations the performance of all the polices for estimating the pose of the robot is statistically similar 3.1.

Real World Integration: For testing out the performance of light curtains with ORB-SLAM in the real world, we outfitted a remote controlled wheelchair with light curtains 3.1. Here we show a qualitative result of the reconstruction of a corridoor environment 3.5. From the figure we see, we are able to faithfully reconstruct the scene.

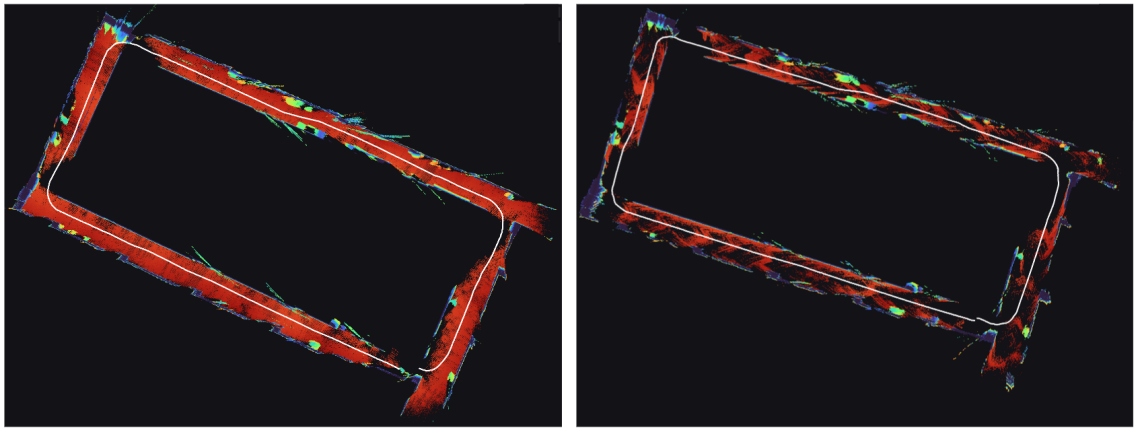


Figure 3.5: Reconstruction result of a corridoor environment.(a) Reconstructed using planar sweeps, (b) Reconstructed using hugging policy. The pointclouds are reconstructed by registering detections from light curtains to a common frame using odometry estimates from ORB-SLAM with light curtains.

	Matterport 1	Forest
Fixed Depth Planar	0.132 ± 0.045	0.222 ± 0.127
Random curtains	0.206 ± 0.188	0.273 ± 0.261
Planar sweeps	0.181 ± 0.07	0.222 ± 0.167
Hugging Policy	0.167 ± 0.067	0.241 ± 0.171

Table 3.1: Performance of ORB-SLAM [6] with Light curtains. The absolute trajectory error (ATE) in meters is average over three runs in each of the environments

	Matterport 2
Fixed Depth Planar	0/5
Random curtains	1/5
Planar sweeps	0/5
Hugging Policy	3/5

Table 3.2: Performance of ORB-SLAM [6] with Light curtains on a trajectory that contains many large rotation. Here we report success (estimating pose over the entire trajectory) over 5 runs.

3.6 Obstacle Avoidance and Path-Planning with light curtains

The other important component to enable autonomous navigation is equipping the robot with obstacle avoidance and path planning module. In this section we describe the process of integrating the estimated odometry and map from light curtains with the autonomy stack of [7].

3.6.1 Integration with Autonomous Exploration Development environment [7]

Terrain Analysis: The main function of the terrain analysis module is to generate a terrain traversability map. It takes as input the current pose of the robot (odometry) and a 3D pointcloud and maintains a volumetric traversability grid around the current position of the robot. This grid is used by successive modules to plan a collision free path to the goal.

In our case, odometry is given by the integration of light curtains with ORB-SLAM as described in the previous sections 3.4. The input 3D pointcloud is computed from the detected points obtained by placing curtains using hugging policy 3.4.2. We use hugging policy since we found to be most robust in our experiments 3.2, 3.4.

Local Planner: The local planner is based on [50]. The input to this module is terrain traversability map and motion primitives. Essentially, motion primitives are the collection of pre-computed paths. Next, we give overview of the working of the local planner. First, terrain traversability map is filtered and cropped to generate the obstacle map for the local planner. Based on this obstacle map, motion primitives are filtered to only contain paths that do not collide with any obstacle. From among the remaining collision free paths, the most likely path to reach the goal is chosen. The chosen path is then used by a **Path Follower** module to generate the command velocities for the motors. The block diagram for the entire process is visualized in 3.6.

Experiments: We tested our integration in an indoor scene on two trajectories. The results are visualized in Figures 3.7.

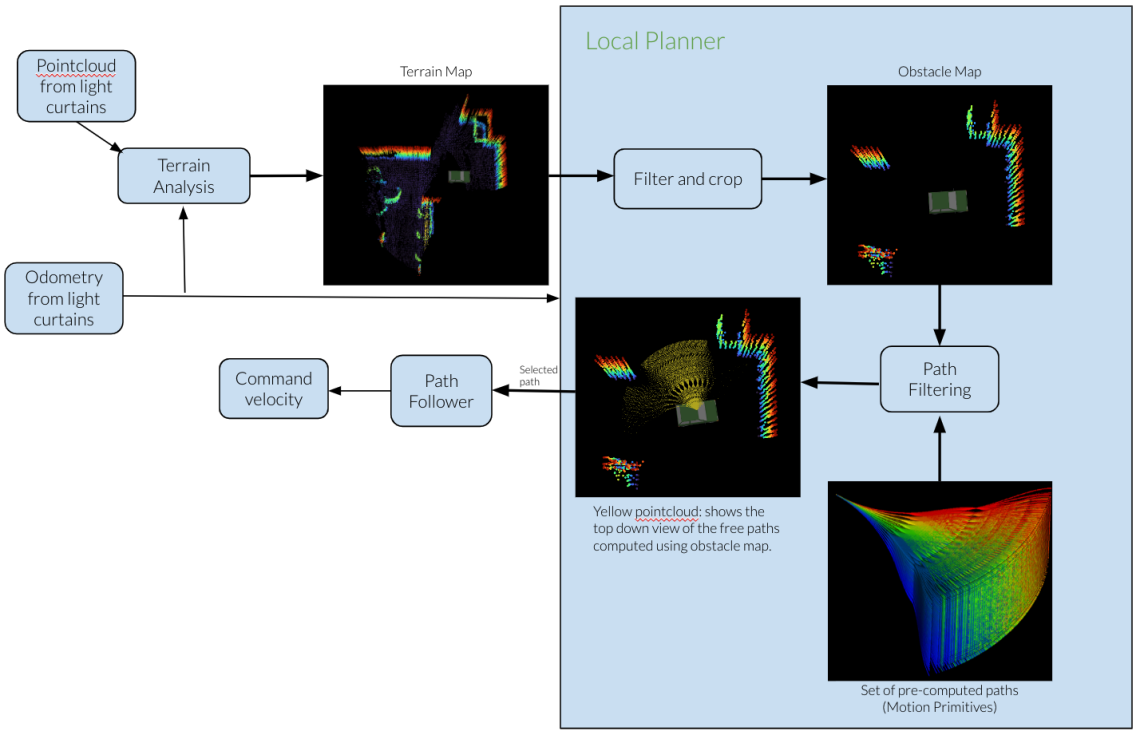


Figure 3.6: Block diagram for integration with [7] is visualized. The input point cloud and odometry to the system is computed from light curtains

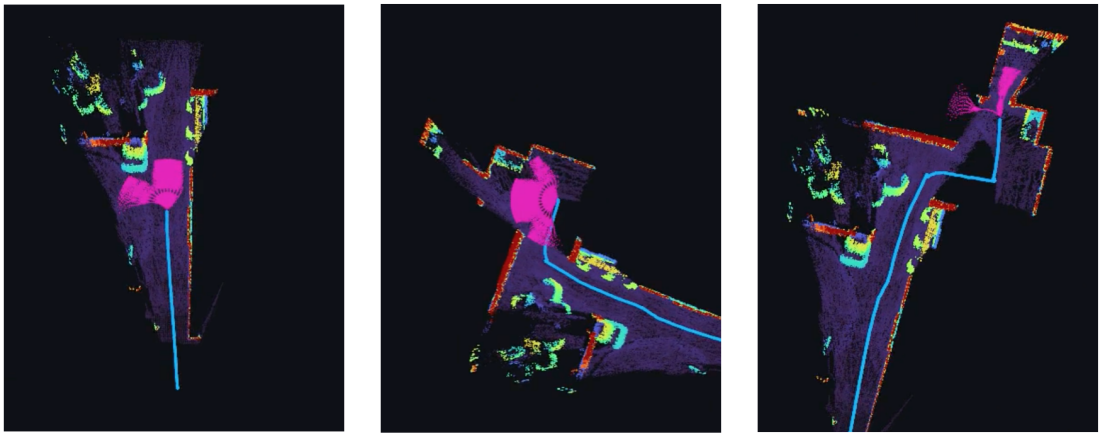


Figure 3.7: Results of integrating light curtains detections and state-estimation with AEDE [7]. (a), (b), (c) visualizes the robot motion at the beginning, middle and the end. Colored pointcloud represents the accumulated terrain map over time. Pink pointcloud shows the free paths. The free paths safely avoid the obstacles.

3.7 Limitations and Future Work

Even though, the hugging policy outperforms other policies when the robot undergoes sharp turns, its success rate is about 60%. Performance under these hard conditions could potentially be boosted by the use of Inertial Measurement Unit. We submit this as one of the future works. Further, we also propose to quantitatively evaluate the accuracy of the reconstructed map by all policies. The system is currently quantitatively evaluated on an indoor environment and an outdoor environment. To further test the robustness of the system we propose to test it on multiple houses from Matterport [8] dataset.

3.8 Conclusion

In this work, we describe the procedure to integrate Programmable Light curtains to enable autonomous navigation on a mobile robot. We first integrate light curtains with state of the art SLAM system ORB-SLAM. We outline the issues related to initialization of odometry with a sparse pointcloud obtained from light curtains. Importantly, we describe the method of using volumetric planar sweeps to resolve this issue. We then outline different policies that we used to get depth for each frame. We test out these policies in different environments and show qualitatively and quantitatively that reconstructed map and odometry obtained from using hugging policies are more robust. Lastly, we combine the odometry and environment pointcloud computed using light curtains with the local planning module of [7] to perform autonomous navigation.

Chapter 4

Conclusions

In conclusions, we first address the problem of tracking the motion of dynamic objects. To reliably track complicated motions of dynamic actors we introduce "Safety Envelopes" and compute them using light curtains. Safety Envelopes are imaginary surfaces that mark the boundary between occupied and unoccupied regions. A robot is guaranteed to not collide with any object in the scene as long as it does intersect with the Safety Envelopes. We use feasible random curtains and previously placed Safety Envelopes in a deep learning based framework to efficiently forecast the Safety Envelope of the scene. Further, we rigorously show the efficacy of random curtains to detect objects in the scene. This framework enables robust robot perception which generalizes to arbitrary number of dynamic objects.

In the second part of the thesis, we work towards enabling full stack autonomous navigation using light curtains. We expound on the process of integrating light curtains with a state of art SLAM method: ORB-SLAM [6]. ORB-SLAM is sensitive to initialization and fails to estimate the ego-motion if initialized with sparse pointcloud. To provide dense pointcloud for initialization we use planar sweeps. We then introduce different polices that are used to provide depth for successive frames. Here we observe that for mostly linear robot motion the performance of all the policies is statistically similar. But for motions involving large number of sharp turns, the so called "hugging policy" outperforms others. Finally, to enable autonomous navigation, we integrate light curtains assisted SLAM and depth detections with the path planning and low level control pipeline of AEDE [7].

Appendix A

Appendix

A.0.1 Transition distributions for sampling random curtains

We first describe, in Algorithm **??** 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 (2.1, 2.2).

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

Algorithm 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 $\leftarrow \{\mathbf{X}_1, \mathbf{X}_2\}$

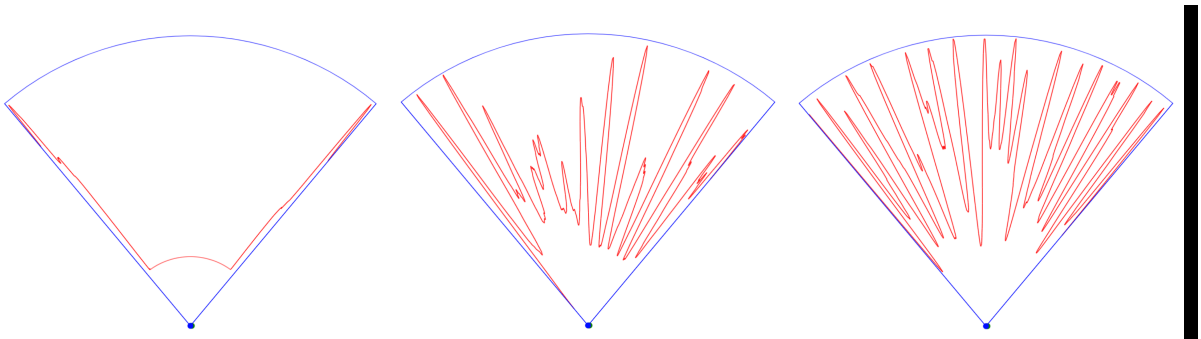
```
/* Iteration                                                 */
```
for $t = 2$ *to* $T - 1$ **do**
  $\mathbf{X}_{t+1} \sim P(\mathbf{X}_{t+1} \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$
  Curtain $\leftarrow$ Curtain $\cup \{\mathbf{X}_{t+1}\}$

return Curtain

and qualitatively evaluate them in terms of the area covered by random curtains generated from them.

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. A.1 (a).

2. Uniform linear setpoint sampling: a more principled way to sample



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

Figure A.1: 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.

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. A.1 (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 $s \sim \text{Uniform}(0, r_{\max}^2)$ 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. A.1 (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.

A.0.2 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* $P_{\text{det}}(\mathbf{X}_{t-1}, \mathbf{X}_t) = P(\bigvee_{t'=t}^{T} \mathbf{D}_t \mid \mathbf{X}_{t-1}, \mathbf{X}_t)$. 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.

A.0.3 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. [1], 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.

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. 2.7.1. 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.

A.0.4 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. A.2. 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 [35] architecture. This involves skip connections between downsampled and upsampled layers with the same spatial

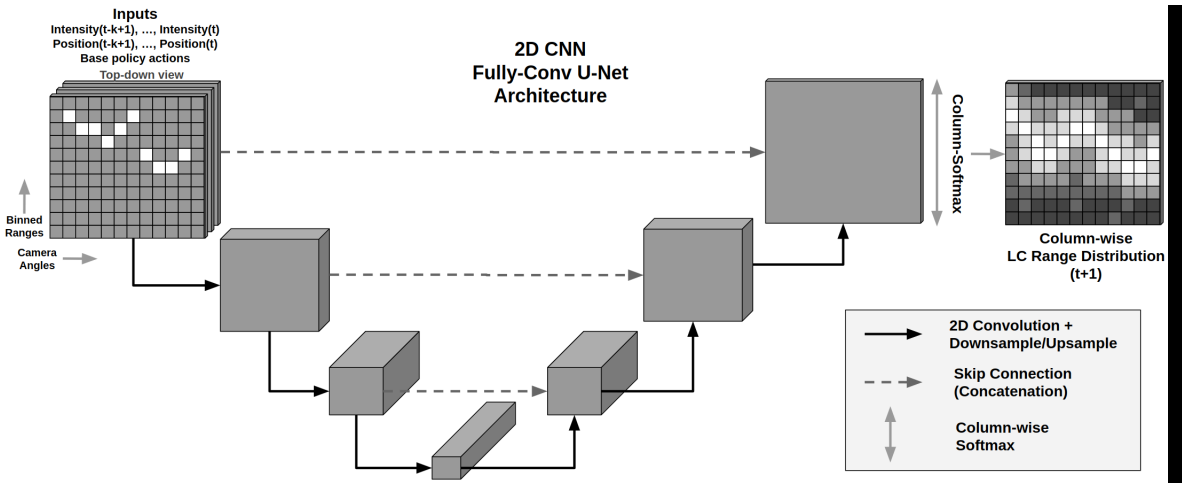


Figure A.2: The network architecture of the 2D CNN model used for safety envelope forecasting. It takes as input the previous k light curtain outputs, and converts them into top-down polar occupancy maps. Each column of the image is assigned to a camera ray, and each row is treated as a binned location. 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 [35] 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.

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.

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 node gets exactly one feature: the intensity value of its corresponding curtain and camera ray. 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.

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 2.1). 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.

A.0.5 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 A.3.

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 A.3 (*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.

A.0.6 Results for the simulated environment without using random curtains

In this section, we include additional results corresponding to the "Ours w/o Random curtains" row of Table 2.1. Table A.1 contains results of other policies (handcrafted baseline, 1D-CNN baseline, 1D-GNN baseline) and ablation conditions (Ours w/o Forecasting, Ours w/o Baseline input) when random curtains are not used. The top half of Table A.1 contains results without using random curtains. The bottom half contains results for the same policies using random curtains (this is essentially a copy of Table 2.1, to aid with comparisions). We find that the conclusions of Table 2.1 still hold when random curtains are not used: our method still outperforms the baselines and removing any component of our method (not forecasting to the next timestep or removing the output of the hand-crafted policy as input) reduces performance.

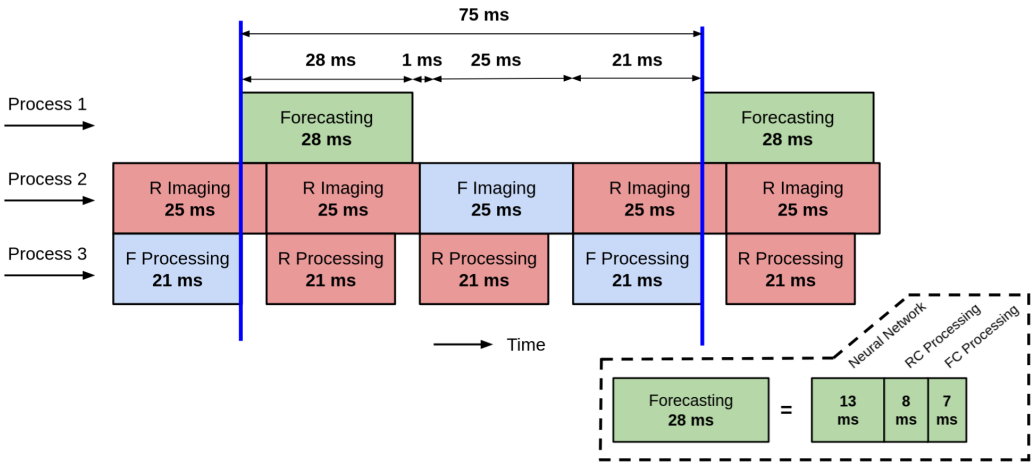


Figure A.3: Pipeline showing the runtime of the efficient, parallelized implementation of our method. The pipeline contains three processes running in parallel: (1) the method that forecasts the safety envelope, (2) imaging of the light curtains performed by the physical light curtain device, and (3) 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.

	Random curtain	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)
		↓	↓	↓	↓	↓	↓	↑	↑	↑
Handcrafted baseline	✗	0.1989	2.5811	0.2040	0.0904	0.2162	2.0308	0.6321	0.7321	0.7657
1D-CNN	✗	0.1522	2.3856	0.2176	0.1076	0.1750	0.9482	0.5842	0.7197	**0.7868**
1D-GNN	✗	0.1584	2.2114	0.1835	**0.0839**	0.1772	1.1999	0.6546	0.7381	0.7710
Ours w/o Forecasting	✗	0.1691	2.6047	0.2288	0.1158	0.1927	1.2555	0.6109	0.7114	0.7654
Ours w/o Baseline input	✗	0.1556	2.5987	0.2273	0.1135	0.1797	1.1063	0.6021	0.7094	0.7683
Ours	✗	**0.1220**	**2.0332**	**0.1724**	0.0888	**0.1411**	**0.9070**	**0.6752**	**0.7450**	0.7852
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	✓	0.0896	1.7124	0.1372	0.0731	0.1101	0.7076	0.7159	0.7900	0.8138
1D-GNN	✓	0.1074	1.6763	0.1377	0.0669	0.1256	0.8916	0.7081	0.7827	0.8037
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 A.1: Performance of safety envelope estimation on the SYNTHIA [53] urban driving dataset under various metrics, with and without using random curtains. Policies in the top half of the table were trained and evaluated without random curtains, while policies in the bottom half were trained and evaluated with random curtain placement.

	Light curtain	Velodyne HDL-64E LiDAR
Horizontal resolution	0.08°	0.08° -– 0.35°
Vertical resolution	0.07°	0.4°
Rotation speed	60 Hz	5 Hz – 30 Hz
Cost	Less than $1000	Approx. $80,000

Table A.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.

A.0.7 Hardware specification of light curtains

In this section, we provide some details about the hardware specification of light curtains, as well as comparing it with the specifications of a Velodyne HDL-64E LiDAR.

The distance between the camera and the laser (the baseline of the device) is 20 cm. The maximum angular velocity of the galvanometer is 2.5×10^4 rad/sec and the maximum angular acceleration of the galvanometer is 1.5×10^7 rad/sec^2. The operating range of the light curtain device is up to 20 meters (daytime outdoors) and 50 or more meters (indoor or night time).

The following table compares the light curtain device with a Velodyne HDL-64E:

The LiDAR is limited to fixed scan patterns. Light curtains are designed to be programmable as long as the curtain profiles satisfy the velocity and acceleration limits. Note that the resolution of the light curtain is the same as the 2D camera used which can be significantly higher than any LIDAR. Our current prototype uses a camera with a resolution of 640×512.

A.0.8 Results for the real-world environment under high latency

The results for the real-world enviroment with walking pedestrians (see Table 2.2 of Section 2.7) were generated using the parallelized and efficient pipeline described in Appendix A.0.5. We now present some older results for the same environment that did not use the efficient implementation. Random curtains were not imaged and processed in parallel with the forecasting method. Instead, these operations were performed

sequentially: we alternated between the forecasting step and placing a single random curtain. This increases the latency of the pipeline. A comparison between our method and the handcrafted baseline when both use this slower implementation is shown in Table A.3. Our method is able to outperform the handcrafted baseline under various implementations with varying latencies.

	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)
	↓	↓	↓	↓	↓	↓	↑	↑	↑
Handcrafted baseline	0.07045	0.7501	0.1282	0.0886	0.1070	0.1072	0.8907	**0.9975**	**1.0000**
Ours	**0.0189**	**0.3556**	**0.0667**	**0.0443**	**0.0449**	**0.0271**	**0.9890**	0.9953	0.9976

Table A.3: Performance of safety envelope estimation in the real-world pedestrian environment under a high latency i.e. slower implementation.

Bibliography

[1] 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. ISBN 978-3-030-58558-7. URL http://siddancha.github.io/projects/active-perception-light-curtains/. (document), 2.1, 1, 2.2.1, 2.1, 2.3, 2.4, A.0.3

[2] 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. URL https://ieeexplore.ieee.org/abstract/document/9030133. 2.2.3

[3] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988. URL https://ieeexplore.ieee.org/abstract/document/5968. 2.1, 2.2.1

[4] Ruzena Bajcsy, Yiannis Aloimonos, and John K Tsotsos. Revisiting active perception. *Autonomous Robots*, 42(2):177–196, 2018. URL https://link.springer.com/article/10.1007/s10514-017-9615-3. 2.1, 2.2.1

[5] 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. URL http://www.cs.cmu.edu/~ILIM/agile_depth_sensing/html/index.html. (document), 1, 2.1, 2.2.1, 2.3, 3.1, 3.1, 3.2, 3.4

[6] Carlos Campos, Richard Elvira, Juan J. Gomez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021. (document), 1, 3.1, 3.2, 3.3, 3.4.1, 3.5, 3.1, 3.2, 4

[7] Chao Cao, Hongbiao Zhu, Fan Yang, Yukun Xia, Howie Choset, Jean Oh, and Ji Zhang. Autonomous exploration development environment and the planning algorithms. In *2022 International Conference on Robotics and Automation*

(ICRA), pages 8921–8928. IEEE, 2022. (document), 1, 3.2, 3.3, 3.5, 3.6, 3.6.1, 3.6, 3.7, 3.8, 4

[8] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. 3.7

[9] 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. URL http://proceedings.mlr.press/v87/cheng18a.html. 2.2.1

[10] Cl Connolly. The determination of next best views. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 432–435. IEEE, 1985. URL https://ieeexplore.ieee.org/abstract/document/1087372. 2.2.1

[11] D Cremers, J Engel, and V Koltun. Direct sparse odometry. *arXiv preprint arXiv:1607.02565*, 2016. 3.2

[12] 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. URL https://openaccess.thecvf.com/content_cvpr_2018_workshops/papers/w9/Kumar_DepthNet_A_Recurrent_CVPR_2018_paper.pdf. 2.2.2

[13] 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. URL https://scholar.google.com/scholar?cluster=7456760468603259697&hl=en&as_sdt=5,39&sciodt=0,39. 2.2.1

[14] 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. URL https://ieeexplore.ieee.org/abstract/document/982896. 2.2.1

[15] 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. URL https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Doumanoglou_Recovering_6D_Object_CVPR_2016_paper.html. 2.2.1

[16] 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. URL https://proceedings.neurips.cc/paper/2014/file/7bccfde7714a1ebadf06c5f4cea752c1-Paper.pdf. 2.7.2

[17] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014. 3.2

[18] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. Georgia Institute of Technology, 2015. 3.2

[19] 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. URL http://www.cvlibs.net/datasets/kitti/. (document), 2.7.1, 2.3

[20] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010. 3.3

[21] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992. URL https://www.jstor.org/stable/2238020?seq=1. 2.6

[22] 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. URL https://ieeexplore.ieee.org/abstract/document/7487527. 2.2.1

[23] Achuta Kadambi, Ayush Bhandari, and Ramesh Raskar. 3d depth cameras in vision: Benefits and limitations of the hardware. In *Computer vision and machine learning with RGB-D sensors*, pages 3–26. Springer, 2014. 1

[24] 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. URL https://link.springer.com/article/10.1007/s11554-013-0386-6. 2.2.1

[25] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011. 3.3

[26] 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. URL https://openaccess.thecvf.com/content_CVPR_2019/papers/Liu_Neural_RGBrD_Sensing_Depth_and_Uncertainty_From_a_Video_Camera_CVPR_2019_paper.pdf. 2.2.2

[27] Larry Matthies, Richard Szeliski, and Takeo Kanade. Depth maps from image sequences1. URL https://www.ri.cmu.edu/pub_files/pub2/matthies_l_1988_1/matthies_l_1988_1.pdf. 2.2.2

[28] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011. 3.2

[29] 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. URL https://arxiv.org/pdf/2001.02613.pdf. 2.2.2

[30] Tong Qin and Shaojie Shen. Online temporal calibration for monocular visual-inertial systems. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3662–3669. IEEE, 2018. 3.2

[31] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4): 1004–1020, 2018. 3.2

[32] Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. 2017. URL https://dspace.mit.edu/handle/1721.1/115978. 2.2.3

[33] 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. URL https://ieeexplore.ieee.org/abstract/document/6907760. 2.2.3

[34] 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. URL https://link.springer.com/chapter/10.1007/978-3-319-60916-4_19. 2.2.3

[35] 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. URL https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28. (document), A.0.4, A.2

[36] 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. URL http://proceedings.mlr.press/v15/ross11a. 2.6

[37] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. doi: 10.1109/ICCV.2011.6126544. 3.3

[38] Davide Scaramuzza and Zichao Zhang. Visual-inertial odometry of aerial robots. *arXiv preprint arXiv:1906.03289*, 2019. 3.2

[39] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018. URL https://www.annualreviews.org/doi/abs/10.1146/annurev-control-060117-105157. 2.2.3

[40] 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. URL https://dl.acm.org/doi/abs/10.1145/641865.641868. 2.2.1

[41] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999. 3.3

[42] 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. URL https://journals.sagepub.com/doi/full/10.5772/58759. 2.2.1

[43] 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. URL http://www.cs.cmu.edu/~ILIM/programmable_light_curtain/html/index.html. 2.1, 2.2.1, 2.3

[44] 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. URL https://openaccess.thecvf.com/content_CVPR_2019/papers/Wang_Recurrent_Neural_Network_for_Un-Supervised_Learning_of_Monocular_Video_Visual_CVPR_2019_paper.pdf. 2.2.2

[45] 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*, pages 1912–1920, 2015. URL https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Wu_3D_ShapeNets_A_2015_CVPR_paper.html. 2.2.1

[46] 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. URL https://openaccess.thecvf.com/content_cvpr_2018/papers/Zhan_Unsupervised_Learning_of_CVPR_2018_paper.pdf. 2.2.2

[47] 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. URL https://openaccess.thecvf.com/content_ICCV_2019/papers/Zhang_Exploiting_Temporal_Consistency_for_Real-Time_Video_Depth_Estimation_ICCV_2019_paper.pdf. 2.2.2

[48] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, pages 1–9. Berkeley, CA, 2014. 3.2

[49] Ji Zhang, Michael Kaess, and Sanjiv Singh. A real-time method for depth enhanced visual odometry. *Autonomous Robots*, 41(1):31–43, 2017. 3.2

[50] Ji Zhang, Chen Hu, Rushat Gupta Chadha, and Sanjiv Singh. Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation. *Journal of Field Robotics*, 37(8):1300–1313, 2020. 3.6.1

[51] Shishun Zhang, Longyu Zheng, and Wenbing Tao. Survey and evaluation of rgb-d slam. *IEEE Access*, 9:21367–21387, 2021. 3.2

[52] ChaoQiang Zhao, QiYu Sun, ChongZhen Zhang, Yang Tang, and Feng Qian. Monocular depth estimation based on deep learning: An overview. *Science China Technological Sciences*, pages 1–16, 2020. URL https://link.springer.com/article/10.1007/s11431-020-1582-8. 2.7.2

[53] 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. URL https://synthia-dataset.net/. (document), 2.7.2, 2.5, 2.6, 2.1, A.1