Exploiting Uncertainty in Triangulation Light Curtains for Object Tracking and Depth Estimation

Yaadhav Raaj

CMU-RI-TR-21-08

May 6, 2021

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee:
Prof. Srinivasa G. Narasimhan, CMU
Prof. Deva Ramanan, CMU
Siddharth Ancha, CMU

Thesis proposal submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics

Abstract

Active sensing through the use of Adaptive Depth Sensors is a nascent field, with potential in areas such as Advanced driver-assistance systems (ADAS). One such class of sensor is the Triangulation Light Curtain, which was developed in the Illumination and Imaging (ILIM) Lab at CMU. This sensor (comprising of a rolling shutter NIR camera and a galvomirror with a laser) uses a unique imaging strategy that relies on the user providing the depth to be sampled, with the sensor returning the return intensity at said location. Prior work demonstrated effective strategies for local depth estimation, but failed to take into account the physical limitations of the galvomirror, work over long ranges, or exploit the triangulation uncertainty in the sensor. Our goal is this thesis is to demonstrate the effectiveness of this sensor in the ADAS space. We do this by developing planning, control and sensor fusion algorithms that consider the device constraints, and exploit the device's physical effects. We present those results in this thesis.

I would like to thank my advisor Srinivas, and various CMU folks including Joe Bartels, Robert Tamburo, Siddharth Ancha, Chao Liu, Dinesh Reddy, Gines Hidalgo for access to their expertise and resources.

Contents

Chapter 1

Introduction

1.1 Background

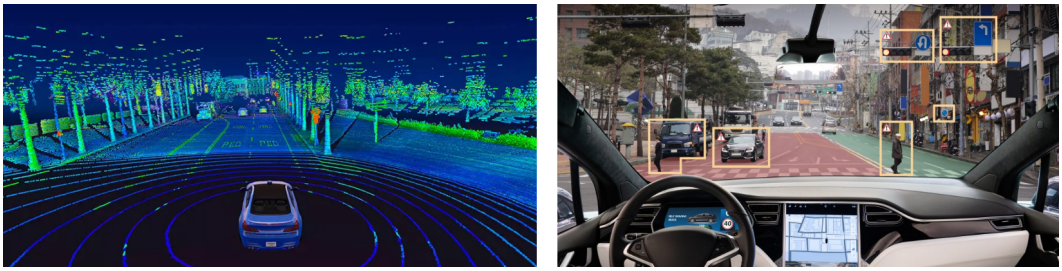

Figure 1.1: **Left:** Advanced Driver-Assistance Systems (ADAS) have become a crucial for vehicular safety, and depth perception forms a core component of it. This is currently achieved through high-cost, high-resolution LIDARs or low-cost, low-resolution RADARs. **Right:** Are we able to exploit the nascent field of low-cost, high-resolution Adaptive Sensors that can sense the depth of regions most important?

The NHTSA (National Highway Traffic Safety Administration) notes that almost 36,000 people died in traffic incidents in the United States [23]. There is currently no law that requires auto manufacturers to install anti-collision hardware and software in their vehicles, and a large majority of vehicles on the road today do not have the appropriate sensors to do so. Newer vehicles manufactured after 2019 have started to install RADARs in their vehicles (eg. Toyota Safety Sense), providing basic collision detection and lane keeping assistance, but these sensors perform poorly in most real-world city-driving scenarios (dense scene with lots of moving objects) due to their low resolution. This makes the current generation of mass installed safety hardware adequate for highway driving only.

On the other spectrum, self-driving companies (Aurora, Argo AI, Waymo) have been using spinning fixed scan LIDARs. These have been the de-factor sensor of choice due to their reliability in depth estimation, but mass adoption is near impossible due to their prohibitive cost. There has been a push towards solid state LIDARs via time-of-flight technology (Ouster, Luminar), but those sensors are yet to show up in real world deployments as of yet. Companies like Tesla and NVIDIA have been using RGB-only depth estimation as

well, but these are nowhere near as reliable as LIDARs, and require immense computational resources and alot of data for training to handle the tail-end of scenarios (low-lighting, fog, oversaturation, scale ambiguity etc.). There has been work that explores how one can capture the uncertainty and error in RGB depth estimation, but there has been limited work in methods that actually exploit and correct this uncertainty.

Finally, we have Adaptive LIDARs. These are sensors that uses various tricks in optics, to achieve a higher depth resolution at the regions that matter most. This is a nascent field, and I only found 3 major approaches that were practical enough to be used in the ADAS space which I will describe in the related work section later. One of these sensors, is the Triangulation Light Curtain [18] [31] which was developed in our lab (ILIM). This is a sensor that allows a user to specify a depth he wishes to measure for a particular row of pixels, and the sensor returns the measured intensity at said location. So can we use this sensor to actually demonstrate practical value in the ADAS space? Before we go further, let us look at a deeper dive at the current state of depth estimation in ADAS with existing sensors.

1.2 Related Work

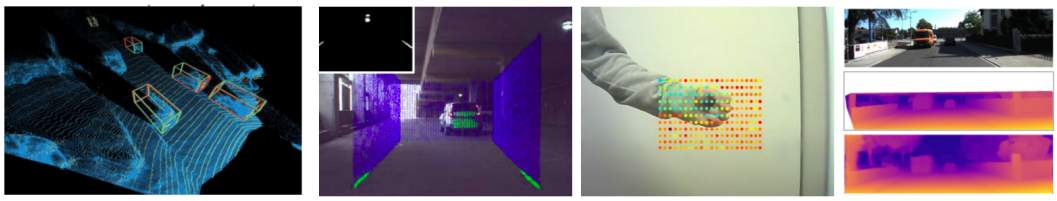

Figure 1.2: **1:** Traditional LIDAR based depth and object detection [32] **2:** Triangulation Light Curtain focusing it's depth on the vehicle ahead [31] **3:** MEMS based Lidar focusing it's depth sensing pixels on the hand of the person [24] **4:** RGB based depth estimation using the KITTI dataset

Depth from Active Sensors: Active sensors use a fixed scan light source / receiver to perceive depth. Long range outdoor depth from these such as commercially available Time-Of-Flight cameras [1] or LIDARs [3] [2] provide dense metric depth with confidence values with wide usage in research [12] [6] [8]. However, apart from low resolution, these sensors are difficult to procure and expensive, making everyday personal vehicle adoption challenging as described earlier.

Depth from Adaptive Sensors: Adaptive sensors use a dynamically controllable light source / receiver instead. The first method is focal length/baseline variation through the use of servos or motors [20] [11] [21] [27]. These are effectively Stereo RGB cameras that have a varying baseline, allowing for the triangulation / stereo ambiguity to be reduced in the regions where the depth is more critical. The second method drives a receiver like a SPAD to pixels (Wetzstein et. al.) [5] [22] [24] or a MEMS mirror and laser to the pixels that matter (Koppal et. al.) [28] [29] [34]. Wetzstein et. al. presents sensors and algorithms for adaptive sensing via 2D angular sampling, providing precise depth at limited number of pixels, but they use a SPAD where light is spread out over the entire FOV limiting it's range and operation outdoors due to ambient light. Koppal et. al. got around this by driving

the light source via MEMS mirrors, but the current hardware implementations have a very limited resolution. The third method uses gated depth imaging [30] [15] [14] where every single pixel of the sensor receives information, but it relies on the user specifying at which slice of depth they want the entire measurement (for all pixel) to be made. If we wanted finer control (specify the depth for each row of pixels), we would have the Triangulation Light Curtain [18] [31]. They increase the adaptability of existing gated depth imaging approaches, and maximize the light energy at the region of interest via triangulation.

Depth from RGB: Depth from Monocular and Multi-Camera RGB has been extensively studied. We focus on a class of Probabilistic Depth estimation approaches that have reformulated the problem as a prediction of per-pixel depth distribution [19] [35] [7] [38] [17] [33]. Some of this work has actually passively exploited and refined [19] [33] the uncertainty in the depth values via Moving Cameras and Multi-View-Camera constraints, but have not used the capabilities of the slew of Adaptive Sensors available.

1.3 Research Problem

It seems like taking a deeper dive into Adaptive Sensors (namely the Triangulation Light Curtain), and whether we can use it practically for ADAS specific problems such as 3D object tracking and depth estimation is a goal worthy of exploring. The existing literature on the Light Curtain (LC) [31] [18] focuses more on the hardware implementations, but has a limited scope on the algorithms and approaches used to solve problems with it.

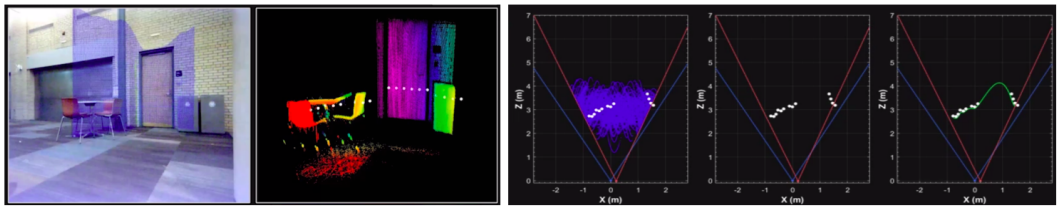

Figure 1.3: Previous work on depth estimation with the Triangulation Light Curtain relied on a sampling strategy that was based on fitting a spline to regions of high intensity to discover the depth in a scene. It didn't take into account the velocity or acceleration constraints of the galvomirror, or take into account the change in intensity as the curtains approached an object.

Previous work on the Triangulation Light Curtain, focused on the sub-problem of depth discovery. It used a baseline method of randomly sampling a B-Spline on a scene. The regions on that spline that had an intensity above a certain threshold was reformulated as a control point for the generation of the next spline. This was then iterated on until depth convergence. This method while fast, had several limitations. Firstly, it did not take into account any of the galvomirror constraints (such as its maximum angular velocity or acceleration), which means that some of the splines generated may not have been imaged correctly. Secondly, it relied on thresholding of the intensity image, which meant that objects with limited returns (such as being far away or being made with a challenging material) did not get their depth imaged. Lastly, it did not use any kind of sensor fusion or use any priors from other sensors in it's approach. If any of these sound confusing, it will become clearer in Section 2.1, which explains the working principle of the Light Curtain sensor.

Hence our goal in this work, is to correctly formalize the problem of 3D object tracking and depth discovery with the Light Curtain. We do this by creating planning and control algorithms that actually take the sensor constraints into account, make use of the actual sensor model with regards to the intensity returns, explore probabilistic approaches to object tracking and depth discovery, and look at how other sensor modalities such as RGB cameras can be used in tandem with it.

The result of this work can be summarized as into the following set of bullet points seen below, and should be reflected in the Table of Contents as well:

- Creating a Simulator for the Light Curtain with tight integration into KITTI and CARLA to facilititate algorithm development and testing
- Developing a Hardware Rig consisting of the Light Curtain, a 128 Beam Lidar, and Stereo Cameras for proper evaluation and testing.
- Coming up with a proper understanding of the Light Curtain's intensity response when imaging objects
- Creating Planning and Control Strategies for Keypoint based imaging, and imaging multiple curtains.
- Creating Planning and Control Strategies for planning a curtain over a discretized space
- Creating a multi object tracking framework with a Particle Filter based approach
- Creating a depth discovery framework based on a Recursive Bayesian update based approach
- Creating a Neural Network that generates a prior from other sensors (RGB cameras), and fusing Light Curtain information into this network

This thesis contains a summary of work found in two published works with my name on it. *Active Perception using Light Curtains for Autonomous Driving* [4] and *Exploiting and Refining Depth Distributions with Triangulation Light Curtains* [26]

Chapter 2

The Triangulation Light Curtain

2.1 Working Principle

Figure 2.1: The Triangulation Light Curtain device, consisting of a steerable laser, an IR camera and a microcontroller, capable of generating a slice in space to image in 3D

Our lab (ILIM) had developed a new kind of sensing technology called a Triangulation Light Curtain [31] [18], a kind of adaptive or steerable LIDAR that allows one to dynamically and adaptively sample the depths of a scene using principles found in stereo based triangulation. The above image demonstrates how a planar curtain is swept across a scene, and when it intersects with the bike, produces an intensity response that can be detected.

The Light Curtain (LC) device consists of a rolling shutter camera flipped vertically, a laser with a galvo-mirror and an RGB camera. The rolling shutter IR camera runs in sync with the laser, where each row of the camera can be thought of as a plane going out vertically into space. The laser/projector fires a similar vertical sheet of light as a plane into space, which intersects with the camera's vertical plane to produce an intersecting line. Any objects that lie within this line will then be imaged by the camera. We we will know the 3D position of said objects (since we know the 3D equation of the intersecting line).

The user/algorithm first decides a path to be traced from a top-down view in the camera frame. We call this set of points P_c. Next, we compute the ray directions of all the pixels p for the middle scanline of the camera from left to right, given the camera intrinsics K. We then interpolate the closest points to P_c that lie on rays r_{norm} in order to produce P_{ci}, which

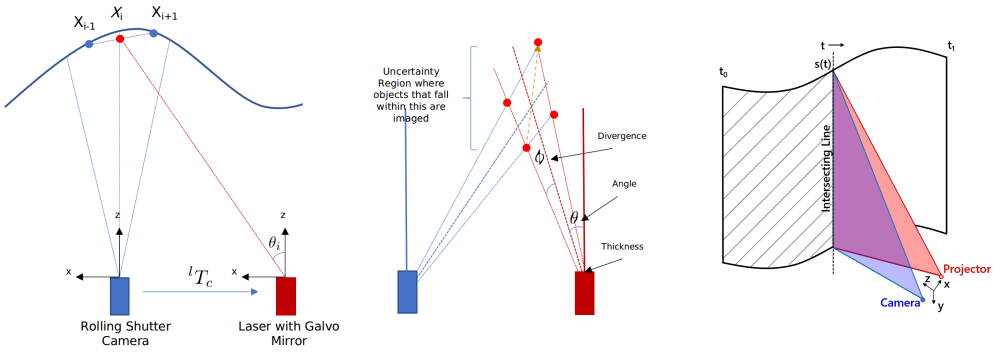

Figure 2.2: The Triangulation Light Curtain device, consisting of a steerable laser, an IR camera and a microcontroller, capable of generating a slice in space to image in 3D. (a) We interpolate control points from the desired curve which can be imaged by the rolling shutter camera. (b) imaging uncertainty inherent in the system. (c) 3D surface imaged by our system

are points that can actually be imaged by our rolling shutter camera.

$$\begin{aligned} r_{norm} &= K^{-1}p \\ P_{ci} &= interpolate(P_c, r_{norm}) \end{aligned} \tag{2.1}$$

We transform these points P_{ci} to the laser frame, and compute the desired angles required by the laser to hit these points. Since our goal is to recover the full 3D surface imaged by the camera and laser, we compute the planes projected by the laser at each of these angles, and then transform the planes back to the camera frame. ${}^{l}T_c$ below denotes the transformation from camera space to laser space.

$$\begin{aligned} P_{li} &= {}^{l}T_c.P_{ci} \\ \theta_{li} &= atan(\frac{P^z_{li}}{P^x_{li}}) + 90 \\ Pl_{ci} &= ({}^{l}T_c)^T.(R_y(\theta_{li})^{-1})^T.[0,0,1,1]^T \end{aligned} \tag{2.2}$$

Finally for each plane Pl_{ci} we iterate over all the camera rays r_{vert} that are exposed while this plane is projected and use plane to line intersection to generate the imaged 3D surface. Let $r_{vert} = (x_i, y_i, 1)$, then

$$\begin{aligned} Ax_it + By_it + Ct + D &= 0 \\ t &= \frac{-D}{Ax_i + By_i + C} \end{aligned} \tag{2.3}$$

which gives us a captured point on the 3D surface. See Figure above for an illustration. In reality however, the camera ray is more of a 3D Pyramid (or a triangle if viewed top down) that goes out into space, and the laser is a cone (or a triangle again if viewed top down) that goes out into space due to it's divergence or thickness. Hence, the intersection results in a volume in space where any objects that intersect it result in higher intensities in the NIR image. This means that as the sensing location approaches the true surface, pixel intensities on NIR image increases. We note that this follows a exponential falloff effect in our later experiment (Fig. 2.3).

We can use the same equations above, and simply model multiple planes and multiple rays in order to compute the thickness of the curtain, and given the true sensing location, we

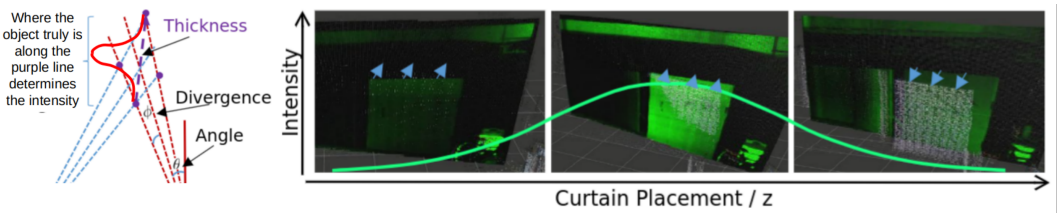

Figure 2.3: The divergence of the camera ray and laser sheet can be described as the curtain *thickness*, which can be thought as a form of triangulation uncertainty. Any object that falls into the region bounded by the quadrilateral will produce some return, which is modelled as an exponential falloff. Note how sweeping a planar curtain across a scene results in a rise and fall of the intensity image

will be able to determine the intensity scaling. The exponential fall-off function used here for each pixel (u, v) is described below. The measured intensity at each pixel is a function of the curtain placement depth $d^c_{u,v}$ on that camera ray, the unknown ground truth depth $\mathbf{d}_{u,v}$ of that pixel, the thickness of the light curtain $\sigma(u, v, d^c_{u,v})$ for a particular pixel and curtain placement, and the maximum intensity possible if a curtain is placed perfectly on the surface $p_{u,v}$ (varies from 0 to 1). From real world data, we find the intensity decays exponentially as the distance between the curtain placement $d^c_{u.v}$ and ground truth depth $\mathbf{d}_{u,v}$ increases, with the scaling factor $p_{u,v}$ parameterizing the surface properties. We also simulate sensor noise as a Gaussian distribution with standard deviation σ_{nse}.

$$i_{u,v} = \exp\left(-\left(\frac{d^c_{u,v} - \mathbf{d}_{u,v}}{\sigma(u, v, d^c_{u,v})}\right)^2\right).p_{u,v} + \sigma_{nse} \tag{2.4}$$

2.2 Simulator

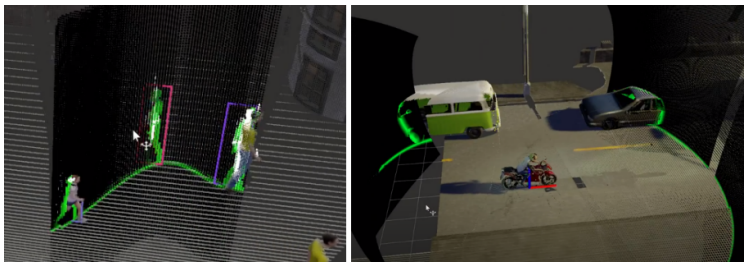

Figure 2.4: The Light Curtain simulator takes in the user designed curtain, and generates a light curtain response similar to what you would find in the real device.

We developed a Light Curtain (LC) simulator using the working principle derived in the previous section. It takes in the NIR camera coefficients (Instrinsics and Distortion), NIR to Laser transformation, Galvomirror constraints, user designed curve, and a depth map. It then outputs the correct intensity value and measured points [XYZI] as a tensor output. This is analogous to the output of the real device. We then integrate this with both CARLA (a ADAS focused simulation environment) and the KITTI dataset (using the upsampled

lidar data). Being able to simulate the LC in KITTI is crucial for our later work. Note that we are unable to simulate the effects on the surface properties $p_{u,v}$ at this time.

2.3 Hardware Setup

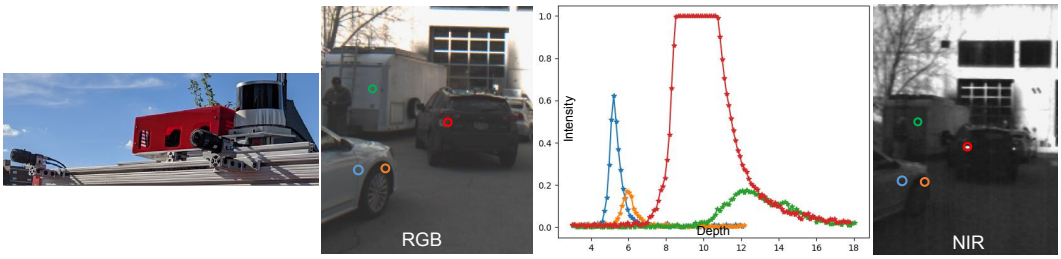

Figure 2.5: **Left:** The hardware and experimental setup consists of a FLIR stereo camera pair with a baseline of 0.7m, the Light Curtain device, and an 128 beam lidar. These are calibrated and mounted onto the CMU jeep. **Right:** We sweep a planar light curtain across a scene at 0.1m intervals, and observe that the changes in intensity over various pixels follow an exponential falloff model. *Blue* / *Orange*: Car door has a higher response than the tire. *Green*: Object further away has a lower response with a larger sigma due to curtain thickness. *Red*: Retroreflective objects cause the signal to saturate.

In order to develop and validate our algorithms, we have a hardware setup that consists of an FLIR RGB stereo camera pair with a baseline of 0.7m, an Ouster OS2-128 beam lidar, and the Light Curtain device itself. We mounted all of these sensors on the jeep, and calibrated the entire setup. The calibration involves computing the instrinsics and distortion coefficients of the NIR on the LC, the extrinsics between the galvomirror and NIR, the galvomirror lookup table that maps the ADC values (-1 to 1) to angles, the intrinsics and extrinsics of the stereo camera pair, and the extrinsics between the lidar and the stereo pair, and the stereo pair and NIR camera.

We then showed that our derivations of the light curtain model are in line with the real device. We swept a planar light curtain across a scene from 3 to 18m at 0.1m intervals. As seen, we note than exponential rise and falloff as the curtain approaches a surface. We also note that some surfaces are more reflective than others, and we note that there is a larger standard deviation in the falloff the further away an object is due to the curtain's thickness.

Chapter 3

Planning and Control

Earlier, we had described how the user needs to input a traced top-down 2D path for the curtain they wish to image via P_c. The current algorithm automatically maps this path to each NIR camera ray via interpolation and handles points that are outside the FOV of both the laser and NIR, to generate a set of points X_c. However, it didn't handle cases where the actual angle and angular velocity / acceleration that the galvomirror had to move were within bounds. Let us look at how we can resolve this.

3.1 Keypoint Based Planning

The simplest task we can solve is to plan a curtain such that the curtain goes through a set of keypoints or control points. There are many cases where the object we want to image is made out of a set of crucial keypoints, but the object itself is rather smooth shaped over a larger volume (eg. a car on a road). Or it could be that we are tracking a set of moving keypoints (eg. a car on the road or people on the road). We could linearly interpolate a curtain across those keypoints, or we could smoothly interpolate it. Let us see the effects it has on the galvomirrors acceleration profile. To see that, we can model the dynamics of the sensor given the points X_c in the camera frame. The angular velocity and the angular acceleration can be described as follows:

$$\begin{gathered} X_l = {}^lT_c.X_c \\ \theta_l = atan(\frac{X_l^z}{X_l^x}) + 90 \\ \dot{\theta} = \frac{d}{dt}(\theta_l) \quad \ddot{\theta} = \frac{d}{dt}\left(\dot{\theta}\right) \quad \delta t = 15us \end{gathered} \tag{3.1}$$

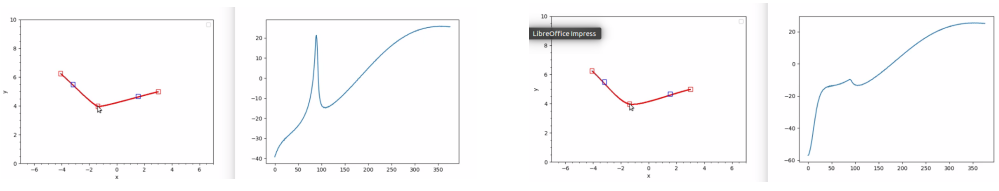

Figure 3.1: $\ddot{\theta}$ before and after smoothing out the path between targets to track

One is able to see that linearly interpolating across the keypoints results in a sharp jerk in the $\ddot{\theta}$ of the galvomirror. We note that smoothing out the curve profile of the middle point via a spline based representation results in a much smoother profile. Let us define a smoothing function that does this. We have chosen to go with Cubic Bezier Curves ([16], [9]), and build upon its formulation to control the smoothness of the curve at the control points.

Bezier curves are a family of B-Splines. However, there is only one polynomial in the piecewise components (The Bernstein polynomials) unlike splines, which can be thought of as the B-Spline basis functions over the domain of those polynomials. We begin by defining a set of control points P with the example below having $P = \{P_0, P_1, P_2\}$. We aim to compute a set of tangent vectors d such that the the curve can be interpolated from the parametric form of cubic beizer curve equation below.

$$B(t) = (P_i)\,(1-t)^3 + (P_i + d_i)\left(3.t.\,(1-t)^2\right) + (P_{i+1} - d_{i+1})\left(3.t^2.\,(1-t)\right) + (P_{i+1})\left(t^2\right) \tag{3.2}$$

We can see from the above example that the following holds true:

$$P_1 - 2.\,(P_1 - d_1) + (P_0 + d_0) = (P_2 - d_2) - 2.\,(P_1 + d_1) + P_1 \tag{3.3}$$

We define the first and last tangent vectors as follows:

$$\begin{array}{cc} A_1 = (P_2 - P_0 - d_0)\,/\alpha_1 & B_2 = -1/\alpha_2 \\ A_i = (P_{i+1} - P_{i-1} - A_{i-1}).B_i & B_i = 1/(\alpha_i + B_{i-1}) \end{array} \tag{3.4}$$

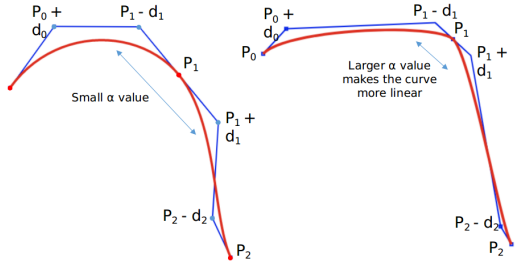

Figure 3.2: Controlling the α parameter makes the curve more linear

Increasing α_i causes the corresponding tangent vector to be shifted closer the control point related to it, making it more linear. We can compute θ from these points by taking the $atan$ of the z and x coordinates at each point, and we approximate $\dot{\theta}$ and $\ddot{\theta}$ via finite differences with $\delta t = 15us$. Using splines in this manner for control can be found in various existing literature. [10], [16], [9], [37].

With this formulation, we simply need to solve for α_i for each control point which we do via numerical gradient descent and simulated annealing. The error term we try to minimize takes the acceleration curve generated, and convolves it with a smoothing ψ_1 and edge detection ψ_2 kernel before taking the overall sum of squares.

$$e = \sigma. \sum_{i=0}^{n} \left(\ddot{\theta} \circledast \psi_1 \circledast \psi_2\right)^2 \tag{3.5}$$

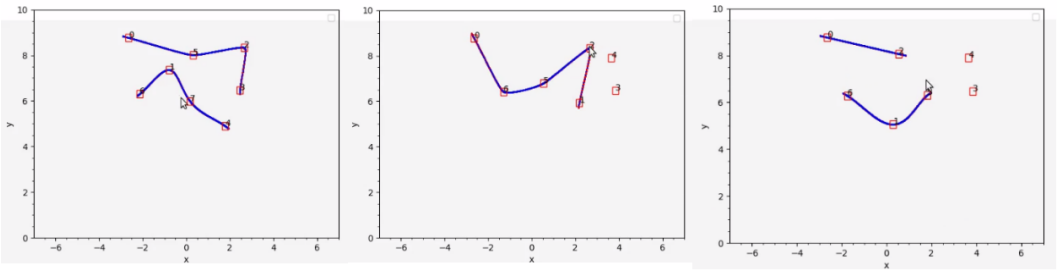

Figure 3.3: Example of splitting and path planning to image all points P. **Left:** All control points can be imaged, but it must be split into 2 curtains being imaged. **Middle:** 2 control points on the right cannot be imaged as they are out of the FOV, but the remaining ones can be imaged in 1 curtain. **Right:** The right 2 control points cannot be imaged as they are out of the FOV, but the remaining can be imaged in 2 curtains.

If a set of generated points are lying close to the same ray, or having a set of points that completely exceed $max(\dot{\theta})$ of the galvo, then we deal with it via splitting operations. We start with $num\_splits = 0$ where we try to select the right ordering of control points P that minimize e. If it exceeds $max(\dot{\theta})$ or if any control points P_i fails to lie near the generated spline X_c, we perform a split, setting $num\_splits = 1$. We try this incrementally where we try to select an ordering and a split such that the following is minimized (where e_i denote the errors for each split):

$$\sum_{i=0}^{num\_splits} (e_i) \tag{3.6}$$

3.2 Imaging Arbitrary Paths

We have a formulation for imaging control points or keypoints, that takes the galvomirror constraints into account, and performs splitting operations to image all points in multiple curtains or passes if required. Can we now extend this formulation to any arbitrary curve?

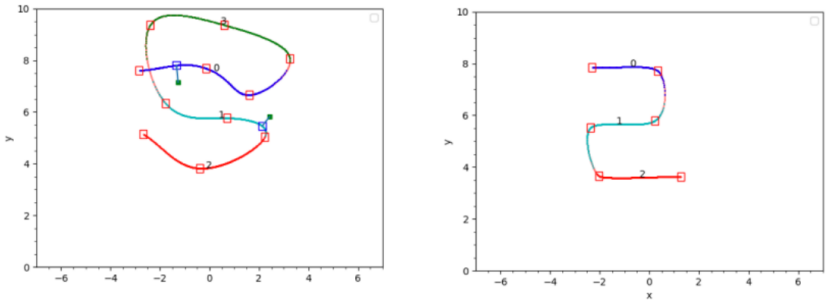

Figure 3.4: Given any arbitrary curve, we have an algorithm that automatically breaks the curve down in such a manner than it can be imaged in it's entirety with the minimum number of light curtain sweeps

Algorithm 1 : Imaging Arbitrary Paths

Input: Arbitrary curve P_c

Output: Multiple curtains needed to image P_c as given by $F = \{F_1 ... F_n\}_c$

procedure IMAGEARBITRARY()
  Initialize empty Subcomponents list $S = \{\}$
  while P_c is not empty **do**
    Image P_c using Interpolation method in Section 2.1
    This generates a subcomponent curve S_i.
    It's guaranteed to be imaged in one light curtain pass. Add S_i to S
    Remove points of S_i from P_c
  end while
  Initialize output list $F = \{\}$
  Initialize combination list $C = \{\}$
  for every i in $MaxSplits = 5$ **do**
    Add the combination of subcomponents $\begin{pmatrix} S \\ k \end{pmatrix}$ to C
  end for
  for every i in $MaxSplits = 5$ **do**
    Iterate C_i / set of curves. Combine the curves.
    Ignore those that can't be imaged ($\dot{\theta} < \dot{\theta}_{max}$)
    Select set of curves in C_i that minimizes e.
    Add it to P and remove combination from rest of C
  end for
end procedure

Our algorithm takes any arbitrary curve P_c, and breaks it down into subcomponent curves that are guaranteed to be imaged in 1 pass by the LC. We then attempt to find a combination of these subcomponent paths that can be imaged when conjoined together, and try to select the smoothest and easiest to image profile using the error term e described earlier. We try to minimize the total number of splits or curtains that need to be imaged this way. With this algorithm, a user can pass any curve and we have a method to image that curve completely. This has benefits when designing complex safety curtains.

3.3 Discretized Planning

The final planning and control method addresses discretized spaces. Let us say that we had a square grid containing the locations in space. We then have each cell in the grid containing a probability from 0 to 1. We wish to plan a light curtain path over this space from left to right that attempts to maximize the cells selected with the highest probability, when ensuring that the path is valid and constraints of the galvomirror are not compromised. This is especially useful when we have these probability grids generated from a neural network or other sensing source. Do note that a large section of this work is from Ancha et. al. [4] of which I am the second author.

Let us consider this discretized probability map U. We must choose control points $\{\mathbf{X}_t\}_{t=1}^{T}$ to satisfy the physical constraints of the light curtain device: $|\theta(\mathbf{X}_{t+1}) - \theta(\mathbf{X}_t)| \leq \Delta\theta_{\text{max}}$ light curtain constraints). The problem is discretized into the grid mentioned above,

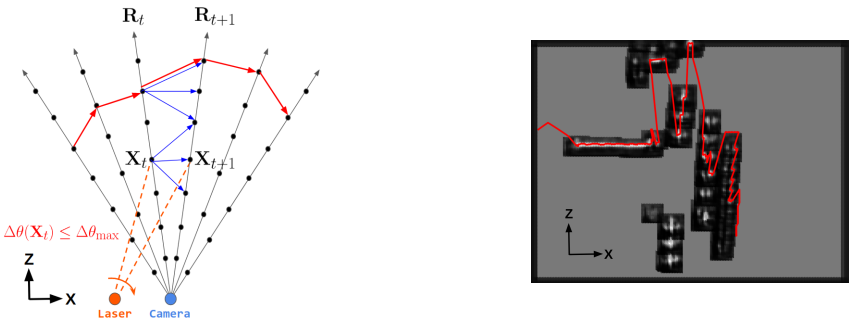

Figure 3.5: (a) Light curtain constraint graph. Black dots are nodes and blue arrows are the edges of the graph. The optimized light curtain profile is depicted as red arrows. (b) Example uncertainty map from the detector and optimized light curtain profile in red. Black is lowest uncertainty and white is highest uncertainty. The optimized light curtain covers the most uncertain regions.

with equally spaced points $\mathcal{D}_t = \{\mathbf{X}_t^{(n)}\}_{n=1}^{N}$ on each ray $\mathbf{R}_t$. Hence, the optimization problem can be formulated as:

$$\arg \max_{\{\mathbf{X}_t\}_{t=1}^{T}} \sum_{t=1}^{T} U(X_t) \quad \text{where } \mathbf{X}_t \in \mathcal{D}_t \; \forall 1 \leq t \leq T \tag{3.7}$$

$$\text{subject to } |\theta(\mathbf{X}_{t+1}) - \theta(\mathbf{X}_t)| \leq \Delta\theta_{\max}, \; \forall 1 \leq t < T \tag{3.8}$$

Assuming we are planning a light curtain across this grid, the total number of possible curtains is $|\mathcal{D}_1 \times \cdots \times \mathcal{D}_T| = N^T$, making brute force search difficult. The problem can be broken down into simpler subproblems however. Let us define $J_t^*(\mathbf{X}_t)$ as the optimal sum of uncertainties of the *tail subproblem* starting from $\mathbf{X}_t$ i.e.

$$J_t^*(\mathbf{X}_t) = \max_{\mathbf{X}_{t+1},\ldots,\mathbf{X}_T} U(\mathbf{X}_t) + \sum_{k=t+1}^{T} U(\mathbf{X}_k); \tag{3.9}$$

$$\text{subject to } |\theta(\mathbf{X}_{k+1}) - \theta(\mathbf{X}_k)| \leq \Delta\theta_{\max}, \; \forall\, t \leq k < T \tag{3.10}$$

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

$$\begin{aligned} J_{t-1}^*(\mathbf{X}_{t-1}) = U(\mathbf{X}_{t-1}) + \max_{\mathbf{X}_t \in \mathcal{D}_t} \; & J_t^*(\mathbf{X}_t) \\ & \text{subject to } |\theta(\mathbf{X}_t) - \theta(\mathbf{X}_{t-1})| \leq \Delta\theta_{\max} \end{aligned} \tag{3.11}$$

This property allows us to solve for $J_{t-1}^*(\mathbf{X}_{t-1})$ via dynamic programming. We also note that the solution to $\min_{\mathbf{X}_1} J_1^*(\mathbf{X}_1)$ is the solution to our original constrained optimization problem (Eqn. 1-3). The recursion from Eqn. 3.11 can be implemented by first performing a backwards pass, starting from T and computing $J_t^*(\mathbf{X}_t)$ for each $\mathbf{X}_t$. Computing each $J_t^*(\mathbf{X}_t)$ takes only $O(B_{\text{avg}})$ time where B_{avg} is the average degree of a vertex (number of edges starting from a vertex) in the constraint graph, since we iterate once over all edges of $\mathbf{X}_t$ in Eqn. 3.11. Then, we do a forward pass, starting with $\arg\max_{\mathbf{X}_1 \in \mathcal{D}_1} J_1^*(\mathbf{X}_1)$ and for a

given $\mathbf{X}_{t-1}^*$, choosing $\mathbf{X}_t^*$ according to Eqn. 3.11. Since there are N vertices per ray and T rays in the graph, the overall algorithm takes $O(NTB_{\text{avg}})$ time; this is a significant reduction from the $O(N^T)$ brute-force solution.

Chapter 4

Sensing and Fusion

In the previous section, we developed algorithms that let us plan curtains that consider the constraints and limitations of the hardware used in the light curtain device. We can use these planning and control strategies in the next section. Here, we explore 3 different ways in which the sensor can actually be used to solve vision problems in the ADAS space. These include multi object tracking (eg. pedestrians on a sidewalk), depth estimation over larger ranges (eg. city driving or parking lots), and fusion with other sensing modalities (eg. RGB, Lidar fusion etc.).

4.1 Multi-Object Tracking

For multi-object tracking, we make use of the Keypoint Based Planning (Sec 3.1). To perform tracking we use one of several Bayesian Filtering approaches, namely a probabilistic particle filter [25]. We begin first by randomly placing curtains in a scene, and attempt to discover tracklets. Let $X_n^{(k)}$ represent a single particle of index k with timestep n, where each particle represents a 3D position (X_s, Y_s, Z_s) and 3D velocity $(\dot{X}_s, \dot{Y}_s, \dot{Z}_s)$. We instantiate $K = 100$ particles per tracklet with a uniform distribution and an equal weight per particle initially, and apply the following motion model with gaussian noise ω added as follows:

$$\mathrm{X}_{n|n-1}^{(k)} = \begin{bmatrix} X_s \\ Y_s \\ Z_s \\ \dot{X}_s \\ \dot{Y}_s \\ \dot{Z}_s \end{bmatrix} \quad X_{n|n-1}^{(k)} = \begin{bmatrix} 1 & 0 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \mathrm{X}_{n-1|n-1}^{(k)} + \omega_{n-1} \tag{15}$$

This lets us generate particles $\mathrm{X}_{n|n-1}^{(k)}$ conditioned on the previous timestep $n-1$. We then apply non-uniform random sampling of the particle weights for each tracklet to generate $M = 3$ keypoints per particle, and use the keypoint based planner to generate $M = 3$ curtains. Fig. 4.1, below shows how this would look like.

We then apply an appropriate measurement model. For example, we could compute weights for each particle k given a measurement Y_n based on a multivariate gaussian dis-

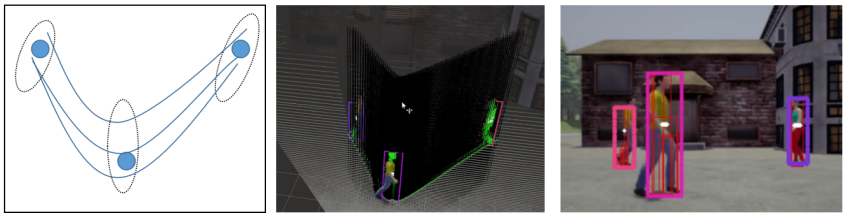

Figure 4.1: In this simulated scenario, we randomly sampled curtains and discovered 3 objects worth tracking. We instantiate 3 particle filters for these 3 objects with $K = 100$ particles with a uniform distribution. We then apply our motion model above, and then apply non-uniform sampling on the particles of each of these 3 tracklets to select $M = 3$ points each (total of 9 keypoints). We then generate $M = 3$ curtains in total that cover these 9 keypoints. The particle weights are recomputed from the signal from these curtain returns as described below

tribution model where d could represent the difference from mean of a particular feature:

$$P\left(\mathrm{Y}_n|\mathrm{X}_{n|n-1}^{(k)}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot exp\left(-\frac{d^2}{2\sigma}\right) \tag{4.1}$$

$$log\left(P(\mathrm{Y}_n|X_{n|n-1}^{(k)})\right) = -log(\sigma\sqrt{2\pi}) - \frac{0.5}{\sigma^2} \cdot d^2 \tag{4.2}$$

We then regenerate K new particles by resampling with replacement given the normalized weight q_n:

$$q_n = \frac{P(\mathrm{Y}_n|\mathrm{X}_{n|n-1}^{(k)})}{\Sigma_k P(\mathrm{Y}_n|\mathrm{X}_{n|n-1}^{(k)})} \quad k \in \mathbb{N} \tag{4.3}$$

$$\mathrm{X}_{n+1|n}^{(k)} = h(\mathrm{X}_{n|n}^{(k)}|q_n) \quad k \in \mathbb{N} \tag{4.4}$$

The measurement model we used is described here. Once these $M = 3$ curtains have been placed, we iterate each particle and minimize 2 objectives. We apply a lower distance cost to particles closer to a curtain placed, and a lower intensity cost to any particle projected back into the NIR camera to get the light curtain intensity returns (Note the exponential falloff model described earlier). We also introduce a collision term to prevent collision between multiple targets (penalizes particles that are too close to another tracklet's particles):

$$q_n^{l2} = -log(\sigma_{l2}\sqrt{2\pi}) - \frac{0.5}{\sigma_{l2}^2} \cdot (l2(Curtain, X_n^{(k)}))^2 \tag{4.5}$$

$$q_n^{int} = -log(\sigma_{int}\sqrt{2\pi}) - \frac{0.5}{\sigma_{int}^2} \cdot ((Proj(X_n^{(k)}))_{int} - 255)^2 \tag{4.6}$$

$$P(\mathrm{Y}_n|\mathrm{X}_{n|n-1}^{(k)}) = q_n^{l2} + q_n^{int} \tag{4.7}$$

We then performed real world experiments in an outdoor setting, varying the number of particles K and number of curtains M sampled from the distribution. As expected, increasing the number of particles results in an improvement in the MOTA (Mean Object Tracking Accuracy), as an increasing number better approximates the underlying distribution of the

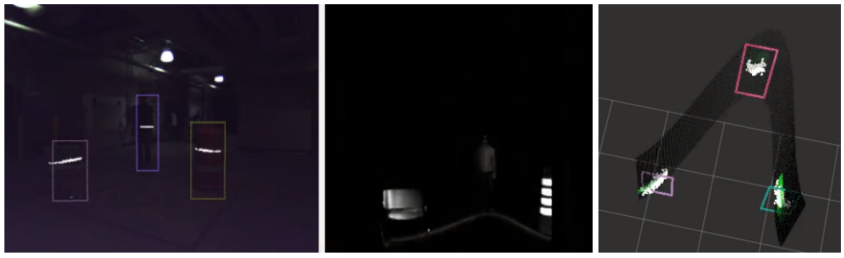

Figure 4.2: The real light curtain sensor tracking multiple targets in real time. It dynamically samples the scene based on the distribution of targets, plans the various galvo positions needed that can optimally track the target, images it, and repeats the process in a closed loop

No. of Particles (3 curtains)	MOTA	fps		No. of Curtains (100 particles)	MOTA	fps
50	48.4	10		1	37.2	28
100	51.4	8		2	48.3	15
150	52.2	7		3	51.4	8
200	52.1	6		4	47.3	4

Table 4.1: In this experiment, we attempted to track the MOTA (Mean Object Tracking Accuracy) of 3 real world objects in a scene. The left table shows that increasing K (number of particles) results in increased tracking performance, up until a certain point. The right table shows that increasing M (number of curtains per tracklet) results in improved tracking performance up until about 3 curtains. Beyond that, the computational and light curtain placement cost results in poorer performance.

object being tracked. However, the improvement does taper off as it is still a function of the number of curtains sampled from the distribution (3 Curtains here). We also did experiments varying the number of curtains sampled from each tracklet. We note improving MOTA scores as we increase the number of curtains, but this is at the expense of the frame rate. At some point, the frame rate drops too much that the tracking accuracy starts to get affected.

While this approach is interesting, it is not tractable computationally with an increasing number of objects. Furthermore, if objects get occluded, it is easy for the particle filter to lose track of the object. This is partially mitigated by the motion model, which works well for the case of pedestrians etc. Also, this tracking problem can be also thought of as a targeted depth estimation problem. In the next line of work, we are interested in seeing if we solve for problem of discovering depth adaptively through the use a bayesian inference formulation.

4.2 Depth Discovery From Light Curtains

4.2.1 Representation

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

$$\mathcal{D} = \{d_0, \ldots, d_{N-1}\}; d_q = d_{\text{min}} + (d_{\text{max}} - d_{\text{min}}) \cdot q \tag{4.8}$$

$$\sum_{q=0}^{N-1} P(\mathbf{d}_{u,v} = d_q) = 1 \text{ (q is the quantization index)} \tag{4.9}$$

$$\text{Depth estimate } = \mathbb{E}[\mathbf{d}_{u,v}] = \sum_{q=0}^{N-1} P(\mathbf{d}_{u,v} = d_q) \cdot d_q \tag{4.10}$$

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

While an ideal sensor could choose to plan a path to sample the full 3D volume, our light curtain device only has control over a top-down 2D profile. Hence, we compress our DPV into a top-down an "Uncertainty Field" (UF) [35], by averaging the probabilities of the DPV across a subset of each column (Fig. 4.3). This subset considers those pixels (u, v) whose corresponding 3D heights $h(u, v)$ are between $(h_{\text{min}}, h_{\text{max}})$. The UF is defined for the camera column u and quantized depth location q as:

$$UF(u, q) = \frac{1}{|\mathcal{V}(u)|} \sum_{v \in \mathcal{V}(u)} P(\mathbf{d}_{u,v} = d_q)$$
$$\text{where } \mathcal{V}(u) = \{v \mid h_{\text{min}} \leq h(u, v) \leq h_{\text{max}}\} \tag{4.11}$$

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

4.2.2 Curtain Planning

We can use the extracted Uncertainty Field (UF) to plan where to place light curtains. We adapt prior work solving light curtain placement as a constraint optimization / Dynamic Programming problem [4]. A single light curtain placement is defined by a set of control points $\{q(u)\}_{u=1}^{W}$, where u indexes columns of the camera image of width W, and $0 \leq q(u) \leq N - 1$. This denotes that the curtain intersects the camera rays of the u-th column at the discretized depth $d_{q(u)} \in \mathcal{D}$. We wish to maximize the objective $J(\{q(u)\}_{u=1}^{W}) = \sum_{u=1}^{W} UF(u, q(u))$. Let $\mathbf{X}_u$ be the 2D point in the top-down view that corresponds to the depth $q(u)$ on camera rays of column u. The control points $\{q(u)\}_{u=1}^{W}$ must be chosen to

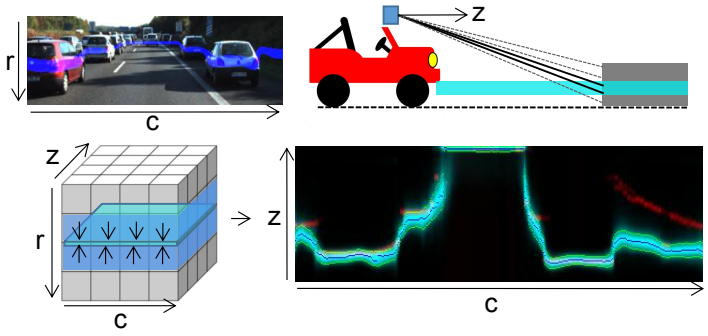

Figure 4.3: Our state space consists of a Depth Probability Volume (DPV) (left) storing per-pixel uncertainty distributions. It can be collapsed to a Bird's Eye Uncertainty Field (UF) (right) by averaging those rays in each row (blue pixels) of the DPV that correspond to a slice on the road parallel to the ground plane (right) (cyan pixels). Red pixels on UF represent the low resolution LIDAR ground truth.

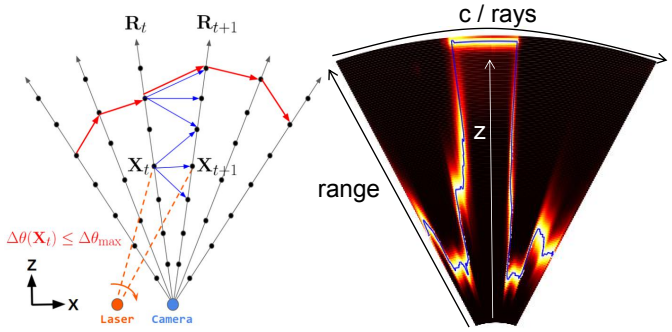

Figure 4.4: Given an Uncertainty Field (UF), our planner solves for an optimal galvomirror trajectory subject to it's constraints (eg. $\mathring{\theta}_{max}$). We show a 3D ruled surface / curtain placed on the highest probability region of UF.

satisfy the physical constraints of the light curtain device: $|\theta(\mathbf{X}_{u+1}) - \theta(\mathbf{X}_u)| \leq \Delta\theta_{\max}$ with $\theta_{\max}$ being the max angular velocity of Galvo:

$$\begin{aligned} &\arg \max_{\{q(u)\}_{u=1}^{W}} \sum_{u=1}^{W} UF(u, q(u)) \\ &\text{subj to } |\theta(\mathbf{X}_{u+1}) - \theta(\mathbf{X}_u)| \leq \Delta\theta_{\max},\ \forall 1 \leq u < W \end{aligned} \tag{4.12}$$

4.2.3 Curtain Placement

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

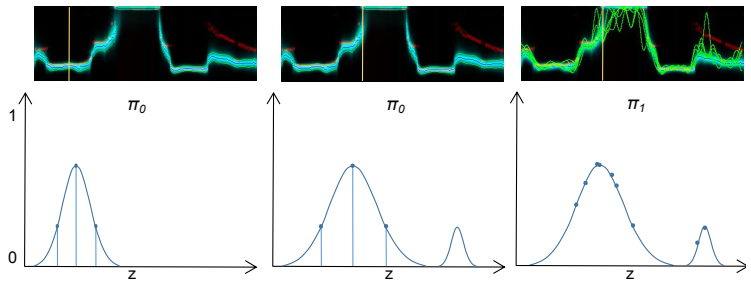

Figure 4.5: Sampling the world at the highest probability region is not enough. To converge to the true depth, we show policies that place additional curtains given UF. Let's look at a ray (in yellow) from the UF to see how each policy works. **Left:** π_0 given a unimodal gaussian with small σ. **Middle:** π_0 given a multimodal gaussian with larger σ. **Right:** π_1 given a multimodal gaussian with larger σ. Observe that π_1 results in curtains being placed on the second mode.

To do this, we propose two policies: π_0 and π_1. In Fig. 4.4, we have placed a single curtain along the highest probability region per column of rays, but our goal is to maximize the information gained. For this, we generate corresponding entropy fields $H(u,q)_i$ to be input to the planner computed from $UF(u,q)$. We use two approaches to generate $H(u,q)$: π_0 finds the mean in each ray's distribution $UF(u)$ and selects a σ_{π_0} that determines the neighbouring span selected. π_1 samples a point on the ray given $UF(u)$.

As seen in Fig. 4.5, strategy π_0 is able to generate fields that adaptively place additional curtains around a consistent span around the mean with some σ_{π_0}, but is unable to do so in cases of multimodal distributions. π_1 on the other hand is able to place a curtain around the second modality, albeit with a lower probability. We will show the effects of both strategies in our experiments.

4.2.4 Observation Model

A curtain placement corresponds to specifying the depth for each camera ray indexed by u from the top-down view. After placing the light curtain, intensities $i_{u,v}$ are imaged by the light curtain's camera at every pixel (u,v). The measured intensity at each pixel is a function of the curtain placement depth $d^c_{u,v}$ on that camera ray, the unknown ground truth depth $\mathbf{d}_{u,v}$ of that pixel, the thickness of the light curtain $\sigma(u,v,d^c_{u,v})$ for a particular pixel and curtain placement, and the maximum intensity possible if a curtain is placed perfectly on the surface $p_{u,v}$ (varies from 0 to 1). This is as what was described in the earlier section on the light curtain return model. As a recap, the sensor model $P(i_{u,v} \mid \mathbf{d}_{u,v}, d^c_{u,v})$ can be described as:

$$P(i_{u,v} \mid \mathbf{d}_{u,v}, d^c_{u,v}) \equiv$$
$$\mathcal{N}\Big(i_{u,v} \mid \exp\Big(-\left(\frac{d^c_{u,v} - \mathbf{d}_{u,v}}{\sigma(u,v,d^c_{u,v})}\right)^2\Big).p_{u,v}, \sigma^2_{\text{nse}}\Big) \tag{4.13}$$

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

4.2.5 Recursive Bayesian Update

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

$$\begin{aligned}
&P_{\text{next}}(u, v, q) \\
&= P(\mathbf{d}_{u,v} = d_q \mid i_{u,v}, d^{c_k}_{u,v}) \\
&= \frac{P(\mathbf{d}_{u,v} = d_q) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})}{P(i_{u,v} \mid d^{c_k}_{u,v})} \\
&= \frac{P(\mathbf{d}_{u,v} = d_q) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})}{\sum_{q'=0}^{N-1} P(\mathbf{d}_{u,v} = d_{q'}) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_{q'}, d^{c_k}_{u,v})} \\
&= \frac{P_{\text{prev}}(u, v, q) \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})}{\sum_{q'=0}^{N-1} P_{\text{prev}}(u, v, q') \cdot P(i_{u,v} \mid \mathbf{d}_{u,v} = d_{q'}, d^{c_k}_{u,v})}
\end{aligned} \tag{4.14}$$

Note that $P(i_{u,v} \mid \mathbf{d}_{u,v} = d_q, d^{c_k}_{u,v})$ is the sensor model whose form is given in Equation 4.13.

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

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

The behavior of this model as the placement depth $d^{c}_{u,v}$, curtain thickness $\sigma(u, v, d^{c}_{u,v})$ and intensity i change is seen in Fig. 4.7. We observe that low intensities lead to an *inverting gaussian* like weight updates, with a low weight at the light curtain's placement location while other regions get uniform weights. This indicates that the method is certain that an object doesn't exist at the light curtain's location, but is uniformly uncertain about the other un-measured regions. A medium intensity leads due a bimodal gaussian, indicating that the curtain may not be placed exactly on the surface and could be on either side of the curtain. Finally, as the intensity rises, so does weight assigned to the light curtain's placement location.

Do note that we have an approximated version of this model, which consists of a aussian and a uniform distribution, where σ is a function of the thickness of the light curtain as described earlier where $t \in [-1..1]$. t is a function of the intensity value i, based on two possible profiles where z either takes a value of 0.5 or 1.0, and m is some control factor we tune:

$$P\left(\mathbf{c}_{it} | \mathbf{d}_t\right) = \frac{\mathcal{N}\left(D_c, \mu_c, \sigma_c\right)(t) + U\left(D_c\right)(1-t)}{\sum\left(\mathcal{N}\left(D_c, \mu_c, \sigma_c\right)(t) + U\left(D_c\right)(1-t)\right)} \tag{4.15}$$

$$t = \left(\frac{-1}{(z) + (mi)}\right) + 1 \tag{4.16}$$

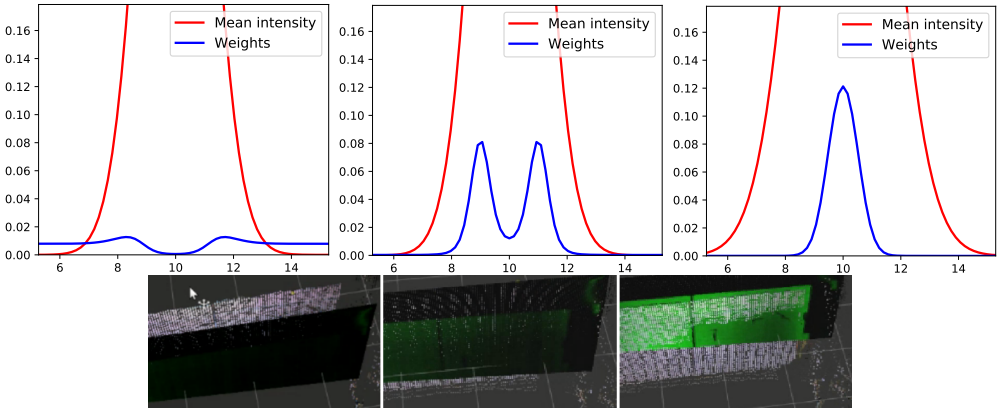

Figure 4.6: Visualization of the recursive Bayesian update method to refine depth probabilities after observing light curtain intensities. The curtain is placed at 10m. The red curves denote the expected intensity (Y-axis) as a function of ground truth depth (X-axis); this is the sensor model given in Eqn. 4.13. After an intensity is observed by the light curtain, we can update the probability distribution of what the ground truth depth might be using our sensor model and the Bayes' rule. The updated probability is shown by the blue curves, computed using the Bayesian update of Eqn. 4.14 (here, the prior distribution P_{prev} is assumed to be uniform, and $d^{c_k}_{u,v} = 10m$). **Left:** Low i return leads to an inverted Gaussian distribution at the light curtain's placement location, with other regions getting a uniform probability. **Middle:** Medium i means that the curtain isn't placed exactly on the object and the true depth could be on either side of the light curtain. **Right:** High i leads to an increased belief that the true depth is at 10m.

The approximated version allows one to test the benefits of the inverting gaussian like effect when the intensity goes to 0.

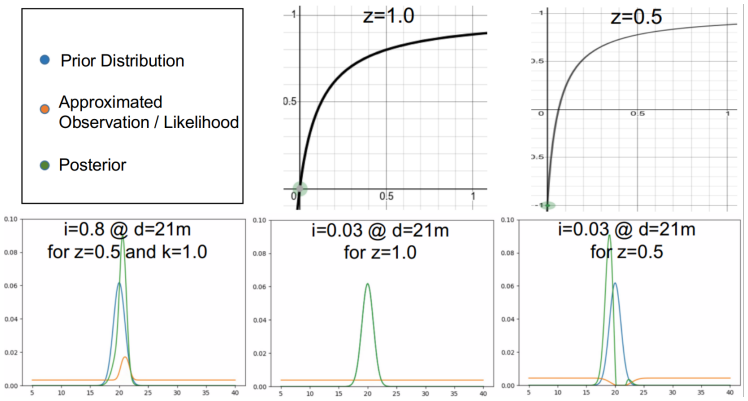

Figure 4.7: Visualizing the approximated observation model. The effect on the posterior when getting a close to 0 intensity return from the Light Curtain when z is either 1.0 or 0.5

4.2.6 Experiments

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

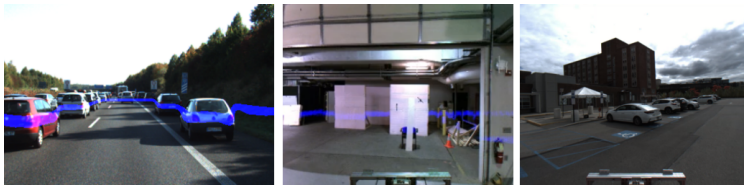

Figure 4.8: Examples of the kind of locations we did experiments in. We were able to evaluate our light curtain only depth discovery algorithms on the KITTI dataset, in the basement with the hardware setup on the NAVLAB Jeep, and in various outdoor scenarios representing various ADAS situations.

Planar Sweep Curtain Placement: We are able to simulate the light curtain response using depth from LIDAR. A simple fixed policy not adapted to the UF helps validate our sensor model and provides corroboration between the simulated and real light curtains. We perform a uniform sweep across the scene above (at 0.25 to 1.0m intervals) (Fig. 4.9), incorporating intensity measurements at each pixel for each curtain using our process described earlier. Our simulated device is able to reasonably match the real device, and we also show how sweeping more curtains increases accuracy at the cost of increased runtime (Table. 4.2).

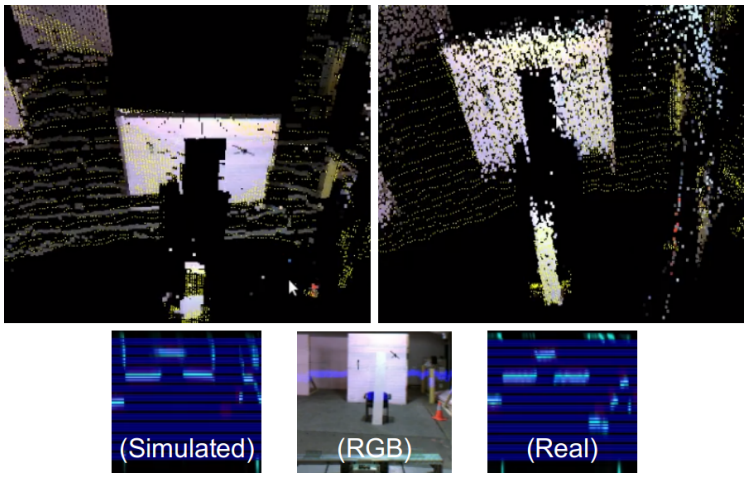

Figure 4.9: We demonstrate corroboration between simulated and real light curtain device by sweeping several planes across this scene. Colored point cloud is the estimated depth, and lidar ground truth in yellow. **Left:** LC simulated from the lidar depth. **Right:** Using the real device.

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

Table 4.2: Policy depicts different numbers of light curtains (LC) placed at regular intervals. The first two columns are simulations and the rest are real experiments. Sampling the scene by placing more curtains results in better depth accuracy (lower RMS) at the cost of higher runtime.

Effect of Dynamic Sigma: Earlier, we had noted how $\sigma(u, v, d^c_{u,v})$ defined for each c_k measurement in $P\left(\mathbf{d}^{c_k}_{u,v}\right)$ is a function of the thickness of the curtain. We also experiment by making $\sigma(u, v, d^c_{u,v})$ fixed. We observe that it being a function of the curtain thickness is critical to better performance over larger steps/placements. Results in Tab. 4.3

Policy	Runtime/s	RMSE/m
Sweep **C** Step 0.25m (Dyn)	2	1.276
Sweep **C** Step 0.5m (Dyn)	1	1.532
Sweep **C** Step 1.0m (Dyn)	0.5	2.013
Sweep **C** Step 0.25m (Fixed)	2	1.218
Sweep **C** Step 0.5m (Fixed)	1	1.658
Sweep **C** Step 1.0m (Fixed)	0.5	2.290

Table 4.3: σ_c in our model $P\left(\mathbf{d}^{c_k}_{u,v}\right)$ being dynamic as a function of curtain thickness as opposed to being fixed, results in better depth estimates

Effect of Inverting Gaussian Model: Our observation model ensures that the sensor distribution tends to an Inverted Gaussian when intensities are low, instead of a Uniform distribution. With our approximated model, we are able to see what happens when a low intensity leads to a uniform distribution (no information) instead. Fig. 4.10 and Fig. 4.11 show significantly improved performance with this inverting gaussian effect.

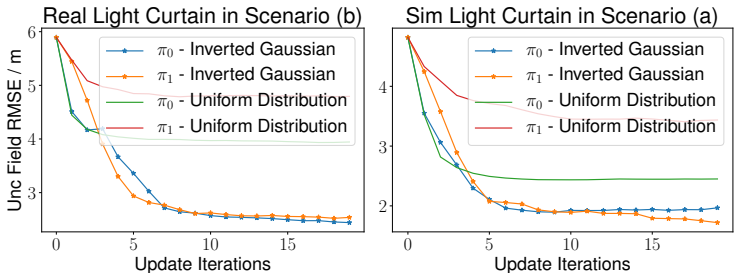

Figure 4.10: RMSE of Depth in UF over every iteration in scenarios left/(a) right/(b). Note the largely improved performance when low intensities tend to an Inverted Gaussian

Policy based Curtain Placement: Sweeping a planar LC can be time consuming (25 iterations), so we want our curtains to be a function of our UF. We evaluated two different scenarios (c1, c2) for each placement policy (π_0, π_1), and we observed that planning and placing curtains as a function of UF results in much faster convergence (Fig. 4.12). We also provide visualizations of what the field looks like in real-world experiments at various iterations (Fig. 4.13).

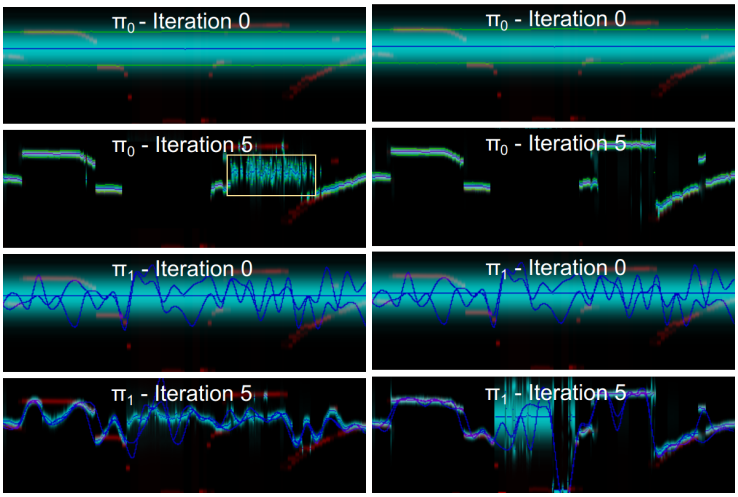

Figure 4.11: **Left:** Policies $m0$ and $m1$ where low intensities result in no information (Uniform Distribution). **Right:** Where low intensities result in an Inverted Gaussian based on our Sensor Model

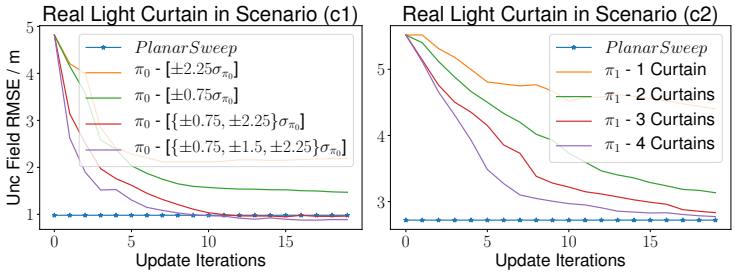

Figure 4.12: Curtain placement as a function of the Uncertainty Field (UF) converges within a lower number of iterations as opposed to a uniform planar sweep which took 25 iterations

Conclusion: We have concluded that this strategy of representing each pixel as a depth distribution (DPV), collapsing said distribution to form an uncertainty field (UF), planning and placing light curtains conditioned on this field, and applying our observation model using a bayesian inference approach is a valid strategy for depth discovery. We are able to validate this in both simulation and in real-world experiments. We have also provided code along with our light curtain sweep dataset so that one may try this process.

Limitations: There are limitations this approach however. For one, the accuracy is limited to the amount of quantization done since the distribution is effectively discretized. We also cannot completely handle surfaces / pixels that return 0 intensity despite being triangulated exactly. Computation is also expensive, and our un-optimized implementation of each planning and curtain placement step takes 40ms. Depth convergence takes about 10 iterations on average (2.5fps). Starting from a gaussian distribution with a large sigma at each timestep means that convergence can take a significant amount of time. Our next goal is to see if we have make use of the temporal nature of driving, or make use of a better prior as an input to our algorithm.

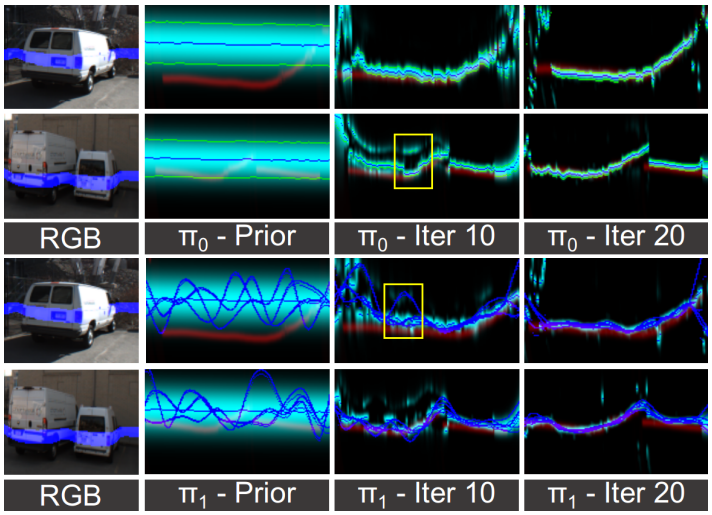

Figure 4.13: Looking at the top-down Uncertainty Field (UF), we see per pixel distributions in Cyan and the GT in Red. We start with a gaussian prior with a large σ, take measurements and apply the bayesian update, trying both policies π_0 and π_1. Note how measurements taken close to the true surface split into a bimodal distribution (Yellow Box)

4.3 RGB + Light Curtain Fusion

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

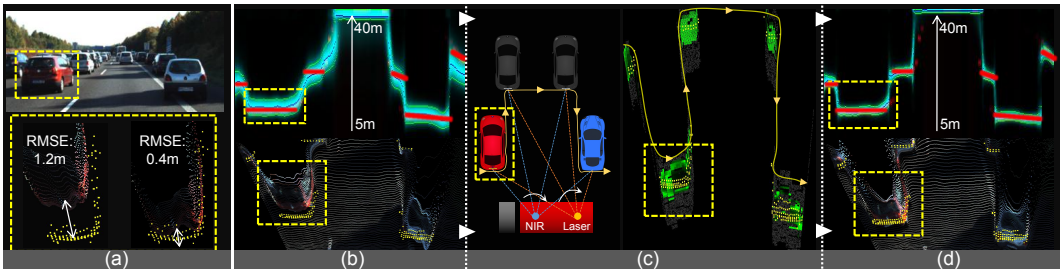

Figure 4.14: (**a**) We show how errors in Monocular Depth Estimation are corrected when used in tandem with an Adaptive Sensor such as a Triangulating Light Curtain (Yellow Points and Red lines are Ground Truth). (**b**) We predict a per-pixel Depth Probability Volume from Monocular RGB and we observe large per-pixel uncertainties ($\sigma = 3m$) as seen in the Bird's Eye View / Top-Down Uncertainty Field slice. (**c**) We actively drive the Light Curtain sensor's Laser to exploit and sense multiple regions along a curve that maximize information gained. (**d**) We feed these measurements back recursively to get a refined depth estimate, along with a reduction in uncertainty ($\sigma = 1m$)

4.3.1 Structure of Network

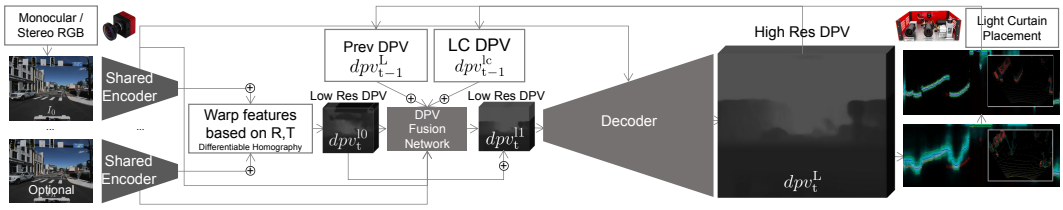

Figure 4.15: Our Light Curtain (LC) Fusion Network can take in RGB images from a single monocular image, multiple temporally consistent monocular images, or a stereo camera pair to generate a Depth Probability Volume (DPV) prior. We then recursively drive our Triangulation Light Curtain's laser line to plan and place curtains on regions that are uncertain and refine them. This is then fed back on the next timestep to get much more refined DPV estimate.

The first step is to build a network (Fig. 4.15) that can generate DPV's from RGB images.

We extend the Neural-RGBD [19] architecture to incorporate light curtain measurements. Anywhere from 1 to N images, usually two (I_0, I_1), are fed into shared encoders, and the features are then warped into different fronto-parallel planes of the reference image I_0 using pre-computed camera extrinsics $R^{I_1}_{I_o}, t^{I_1}_{I_o}$. Further convolutions are run to generate a low resolution DPV $dpv^{\text{l0}}_{\text{t}}$ [H/4, W/4] where the log softmax operator is applied and regressed on. The transformation between the cameras acts as a constraint, forcing the feature maps to respect depth to channel correspondence. The add operator into a common feature map is similar to adding probabilities in log-space.

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

4.3.2 Loss Functions

Soft Cross Entropy Loss: We build upon the ideas in [36] and use a soft cross entropy loss function, with the ground truth LIDAR depthmap becoming a Gaussian DPV with σ_{gt} instead of a one hot vector. This way, when estimating $\mathbb{E}\left(dpv^{gt}\right)$ we get the exact depth value instead of an approximations limited by the depth quantization $\mathcal{D}$. We also make the quantization slightly non-linear to have more steps between objects that are closer to the camera:

$$l_{sce} = \frac{-\sum_i \sum_d \left(dpv^{\{\text{l0,l1,L}\}} * log\left(dpv^{gt}\right)\right)}{n} \tag{4.17}$$

$$\mathcal{D} = \{d_0, \ldots, d_{N-1}\}; d_q = d_{\text{min}} + (d_{\text{max}} - d_{\text{min}}) \cdot q^{pow} \tag{4.18}$$

L/R Consistency Loss: We train on both the Left and Right Images of the stereo pair whose Projection matrices P_l, P_r are known [13]. We enforce predicted Depth and RGB consistency by warping the Left Depthmap into the Right Camera and vice-versa, and minimize the following metric:

$$D_l = \mathbb{E}\left(dpv^L_l\right) \qquad D_r = \mathbb{E}\left(dpv^L_r\right) \tag{4.19}$$

$$l_{dcl} = \frac{1}{n}\sum_i \left(\frac{\left|D_{\{l,r\}} - w\left(D_{\{r,l\}}, P_{\{l,r\}}\right)\right|}{D_{\{l,r\}} + w\left(D_{\{r,l\}}, P_{\{l,r\}}\right)}\right) \tag{4.20}$$

$$l_{rcl} = \frac{1}{n}\sum_i \left(||I_{\{l,r\}} - w\left(I_{\{r,l\}}, D_{\{l,r\}}, P_{\{l,r\}}\right)||_1\right) \tag{4.21}$$

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

$$l_s = \frac{1}{n}\sum_i \left(\left|\frac{\partial I}{\partial x}\right| e^{-|S_x I|} + \left|\frac{\partial I}{\partial y}\right| e^{-|S_y I|}\right) \tag{4.22}$$

The L/R Consistency and Edge aware losses are only applied to the final $dpv^{\text{L}}_{\text{t}}$. While the Soft Cross Entropy Loss is applied to $dpv^{\text{l0}}_{\text{t}}$, $dpv^{\text{l1}}_{\text{t}}$, $dpv^{\text{L}}_{\text{t}}$.

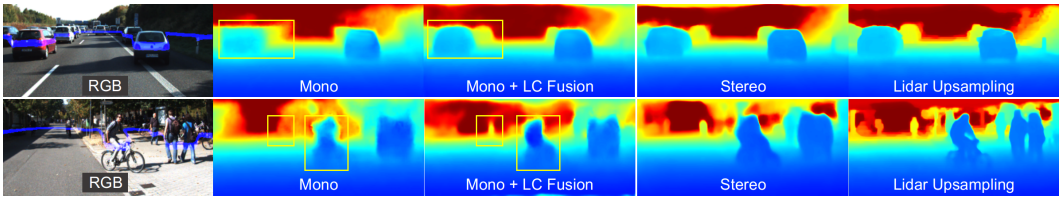

Figure 4.16: In KITTI + Simulated Light Curtain, we note improved depthmaps when Monocular inputs are fused with Light Curtain inputs. Note the improvements in regions bounded in the yellow box. Our network is also capable of ingesting Stereo inputs, and also solving the task of Lidar Upsampling

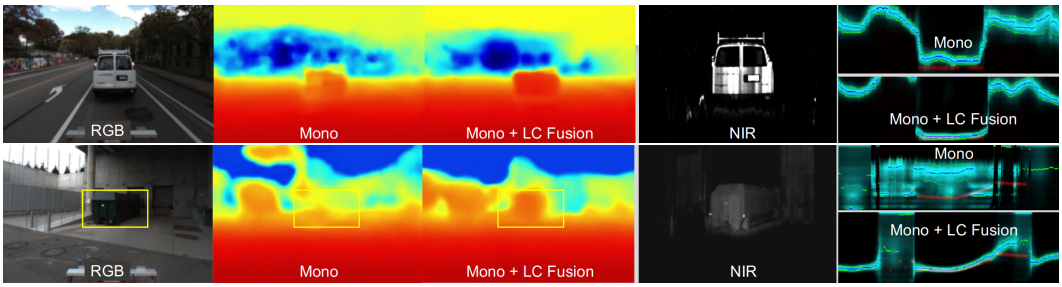

Figure 4.17: In real world Experiments, we are able to see the monocular scale ambiguity in domain specific scenarios (driving scenario with a van 8m away) get corrected by the Light Curtain, and we are able to see correction in an arbitrary scene (dumpster 15m away) provided to the system as well

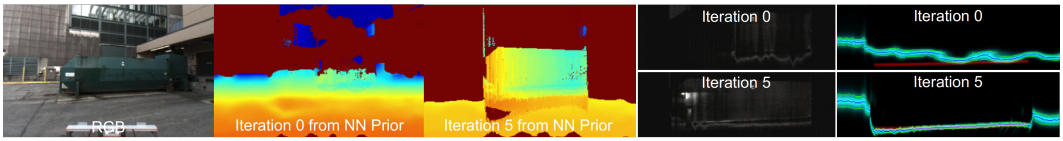

Figure 4.18: We show the internal state of the bayesian update at Iteration 0 and Iteration 5. Starting with a prior DPV from Monocular Depth estimation, we show the convergence of the sensor's laser and curtain profile on an object 10m away

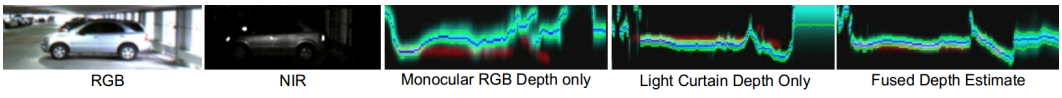

Figure 4.19: Monocular RGB alone suffers from scale ambiguity but does give an inital uncertain depth estimate on a car 15m away. Iterating on Light Curtain measurements from a mean-centered gaussian prior alone gives a more accurate depth but with a noisy profile, but starting with the RGB DPV results in a more accurate and smoother profile.

4.3.3 Light Curtain + RGB Fusion Experiments

We train and validate our algorithms on the KITTI dataset. We then trained the same network by initializing on those weights, but using our custom dataset to evaluate our algorithms with the real sensors on the Jeep. For evaluation, we consider the RMSE metric against the entire depthmap as opposed to just the Uncertainty Field (UF) as $\sqrt{\sum \frac{(\mathbb{E}(\mathbf{d}_{u,v}) - \mathbf{d_{gt}}(u,v)_i)^2}{n}}$ against our ground truth.

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

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

Table 4.4: Effects of Soft Cross Entropy (σ_{gt}), Left/Right Consistency (l_{dcl}, l_{rcl}), Smoothness losses (l_s) on Monocular Depth Estimation.

Table 4.4 shows successively improving performance as we increase σ_{gt}, with poorer performance when the depth is effectively encoded as a one-hot vector (eg. $\sigma_{gt} = 0.05$), since the depth was more likely to be forced into one of the categories in $\mathcal{D}$. Adding in l_{dcl}, l_{rcl} and l_s improved performance further.

Stereo Inputs: Since our method can generalize to any N camera setup, we compare and contrasted monocular pair inputs at times $t, t-1$, against a stereo pair at time t as input (extrinsics known in both cases). As expected, we note significantly better performance with stereo input (Table 4.5) and Fig. 4.20.

Mono vs Stereo

Mono	2.904
Stereo	1.737

Lidar Upsample with DPV Fusion Network

Without DPV Fusion Network	1.118
With DPV Fusion Network	0.702

Table 4.5: **Left:** Stereo pair at t instead of Monocular pair at $t, t-1$ input to the network. **Right:** Fusing the GT LIDAR data with dpv_t^{l0} to generate dpv_t^{l1} and dpv_t^{L} with Bayesian inference vs DPV Fusion Network.

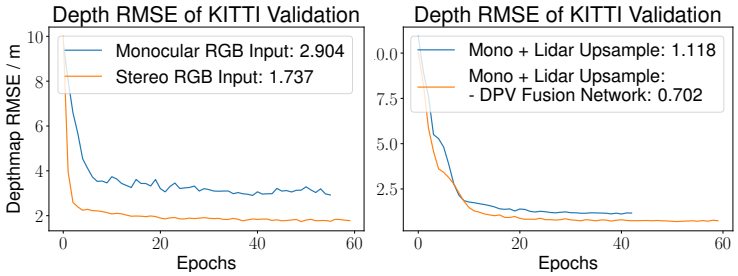

Figure 4.20: **Left:** Stereo pair at t instead of Monocular pair at $t, t-1$ input to the network. **Right:** Fusing the GT LIDAR data with dpv_t^{l0} to generate dpv_t^{l1} and dpv_t^{L} with Bayesian inference vs DPV Fusion Network.

Effect of a Stronger Prior: Previously, we had run our adaptive sensing algorithm from a gaussian prior with a large σ (Fig. 4.11). In various outdoor experiments, we show that a prior DPV from our network instead, yields higher accuracy and faster convergence towards the true depth (Fig. 4.19, Fig. 4.21)(a, b).

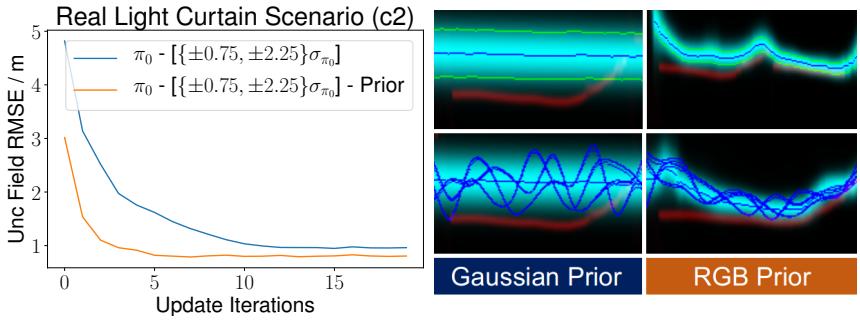

Figure 4.21: Top: Adaptive depth sensing with the Light Curtain: Starting from a Prior distribution from a Monocular Depth Network as opposed to a gaussian with a large σ leads to faster convergence towards the true depth.

DPV Fusion Network: With this corrected DPV, we want to explore how to effectively handle erroneous measurements (due to low light curtain returns etc.), or fuse it other DPVs (from previous frame or from another sensor). With this in mind, we consider the sub task of LIDAR Upsampling. The Velodyne LIDAR in the KITTI dataset, can be converted into a low resolution depthmap, and consequently a low-res DPV we call dpv_t^{gt}. We could then fuse both dpv_t^{l0} and dpv_t^{gt} to generate dpv_t^{l1} using Bayesian inference. Alternatively, we could feed both of those inputs into our DPV Fusion Network, which relies on a series of 3D Convolutions. We note improved performance in this upsampling task using this approach as seen in Table 4.5 and Fig. 4.20.

Light Curtain Fusion Network: Finally, we combine all of these concepts into one. Here, we train our monocular and stereo depth estimation without light curtain feedback, and one where we enable dpv_t^{lc} to be planned and fed-back on the next stage via our DPV Fusion Network, as described in (Fig. 4.15). Training is done on the KITTI dataset with our light curtain simulator, with a maximum of 5 update iterations for performance and memory reasons. In order to perform real world qualitative experiments, we then trained the network (using the weights trained on the KITTI dataset) with the light curtain simulator on our large scale ILIM dataset (similar to KITTI). We observed qualitative (Fig. 4.17) and quantitative (Fig. 4.22) performance improvement of depth with Monocular input, and marginal but visible improvement with Stereo. This is likely due to the wide baseline of 0.7m in the stereo pair, so we could see that smaller baseline pairs would benefit more with our light curtain measurements.

Performance: As mentioned earlier, our un-optimized implementation of each planning and curtain placement step takes 40ms. Depth convergence occurs in 5 iterations (5 fps) when starting from a monocular RGB prior and 10 iterations (2.5 fps) with a Gaussian prior. In temporally continuous operations, the prior from $t-1$ reduces convergence to 2 iterations (12.5 fps) depending on the camera motion. A well-engineered implementation could achieve 20-40 fps but much faster motion would require explicitly encoding 3D optical flow. The network itself takes about 20ms to run it's inference, and since it requires 2 passes

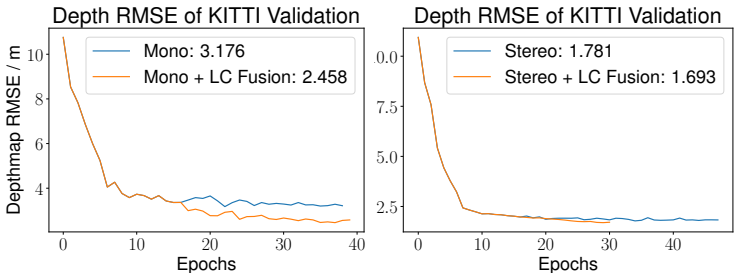

Figure 4.22: Monocular (Left) and Stereo (Right) Depth Estimation show improvements when we enabled feedback of the sensed Light Curtain DPV at epoch 16 when training on KITTI dataset with light curtain simulator.

to ingest the light curtion DPV, it takes 40ms. Furthermore, more work could be done to better explore varying baselines with varying discretization of the DPV to see if there can be greater performance improvements.

Chapter 5

Dataset and Code

5.1 Dataset

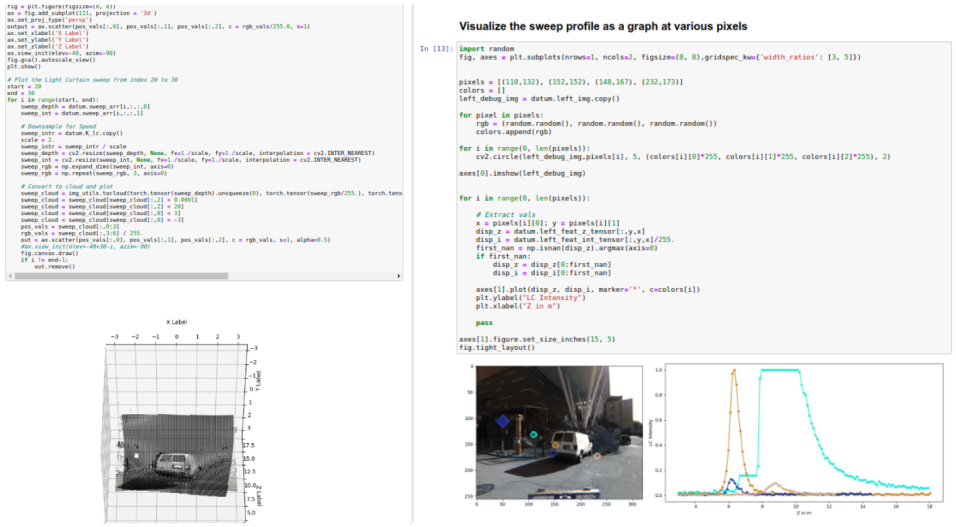

Figure 5.1: A notebook demonstrating what our LC sweep dataset consists of

In order to facilitate work on the algorithm offline, and to give others who do not have access to a Light Curtain device access to real world data, we collected a LC Sweep dataset on various outdoor scenes. This dataset includes Stereo RGB data, ambient NIR data, 128 beam Lidar data, and a planar light curtain sweep volume done from 3m to 18m (at 0.1m intervals). This allows one to "sample" the sweep volume to generate any curtain required, effectively simulating real world light curtain placement in a static scene. We did this on over 600 unique scenes, with about 20 percent of the scenes being temporally consistent (back to back).

Currently, we are only using the dataset for simply evaluating the algorithm we have for depth discovery, which we demonstrate in various notebooks in our code. As noted earlier however, our code is unable to handle cases where the LC return saturates due to retro-

reflective objects, or simply has little to no return due to it's surface properties. It would be possible to use this dataset to train a model that could ingest this ambient NIR or stereo data to learn to predict the $p_{u,v}$ or $\sigma(u, v, d^c_{u,v})$ terms correctly.

5.2 Code

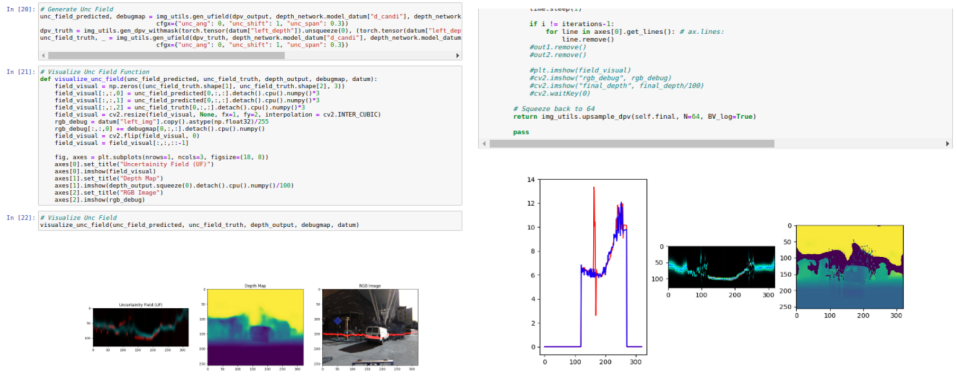

Figure 5.2: A notebook demonstrating the depth discovery algorithm described earlier run on the dataset we collected

The code for downloading the dataset, and for evaluating the depth discovery algorithm can be found at (https://github.com/soulslicer/rgb-lc-fusion-code).

Chapter 6

Conclusion

6.1 Major Findings

We have demonstrated the first known work that has leveraged uncertainty in RGB-based depth estimation to drive an Adaptive Sensor such as a Light Curtain, in the context of ADAS. Our approach *can generalize* to any sensor that uses the principle of driving a laser or light source to specific pixels that are uncertain, and can benefit from depth uncertainty information at a pixel. Through this work, we have created a simulator for our light curtain device, and we have collected large scale datasets for real world experimentation with our hardware setup. We have also created planning and control strategies for curtain placement, subject to the device constraints. Finally, we have demonstrated a multi object tracking framework, and a depth discovery framework based on various bayesian filtering techniques.

6.2 Future Work

One of the biggest issues right now is that normally non-incident and low reflectively surfaces (dark surfaces etc.) result in poor intensity returns. This is intended to be handled by the $p_{u,v}$ in our model, which would tend the generated distribution to a uniform one if the surface is unable to give light curtain returns. We have attempted to use the ambient NIR image to predict this value, but we note that this can be very noisy and only works well if the scene is lit up by a general NIR light source (indirect sunlight). Since we have collected a large scale dataset of these light curtain sweeps, it may be possible to build a model that ingests the RGB and Ambient NIR image to predict the $p_{u,v}$ term for each pixel. Lastly, we could also consider modelling optical flow / scene flow in order to handle temporally changing scenes better (eg. fast moving vehicle).

Bibliography

[1] Luminar. https://www.luminartech.com/.

[2] Ouster. https://ouster.com/.

[3] Velodyne. https://velodynelidar.com/.

[4] S. Ancha, Y. Raaj, P. Hu, S. G. Narasimhan, and D. Held. Active perception using light curtains for autonomous driving. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 751–766, Cham, 2020. Springer International Publishing.

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

[6] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving, 2020.

[7] J.-R. Chang and Y.-S. Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[8] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3d tracking and forecasting with rich maps, 2019.

[9] L. Chen, S. Wang, H. Hu, and K. McDonald-Maier. Bézier curve based trajectory planning for an intelligent wheelchair to pass a doorway. In *Proceedings of 2012 UKACC International Conference on Control*, pages 339–344. IEEE, 2012.

[10] M. Elbanhawi, M. Simic, and R. N. Jazar. Continuous-curvature bounded trajectory planning using parametric splines. In *IDT/IIMSS/STET*, pages 513–522, 2014.

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

[12] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[13] C. Godard, O. M. Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency, 2017.

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

[15] T. Gruber, F. Julca-Aguilar, M. Bijelic, W. Ritter, K. Dietmayer, and F. Heide. Gated2depth: Real-time dense lidar from gated images, 2019.

[16] L. Han, H. Yashiro, H. T. N. Nejad, Q. H. Do, and S. Mita. Bezier curve based path planning for autonomous vehicle in urban environment. In *2010 IEEE Intelligent Vehicles Symposium*, pages 1036–1042. IEEE, 2010.

[17] E. Ilg, Özgün Çiçek, S. Galesso, A. Klein, O. Makansi, F. Hutter, and T. Brox. Uncertainty estimates and multi-hypotheses networks for optical flow, 2018.

[18] W. Joe Bartels, Jian Wang and S. G. Agile depth sensing using triangulating light curtains. In *2019 IEEE International Conference on Computer Vision (ICCV)*, 2019.

[19] C. Liu, J. Gu, K. Kim, S. Narasimhan, and J. Kautz. Neural rgb-¿d sensing: Depth and uncertainty from a video camera, 2019.

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

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

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

[23] NSC, 2020.

[24] F. Pittaluga, Z. Tasneem, J. Folden, B. Tilmon, A. Chakrabarti, and S. Koppal. Towards a mems-based adaptive lidar, 10 2020.

[25] Y. Raaj, A. John, and T. Jin. 3d object localization using forward looking sonar (fls) and optical camera via particle filter based calibration and fusion. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–10, 2016.

[26] Y. Raaj, S. Ancha, R. Tamburo, D. Held, and S. G. Narasimhan. Exploiting and refining depth distributions with triangulation light curtains. 2021.

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

[28] Z. Tasneem, D. Wang, H. Xie, and K. Sanjeev. Directionally controlled time-of-flight ranging for mobile sensing platforms. 06 2018.

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

[30] S. Walz, T. Gruber, W. Ritter, and K. Dietmayer. Uncertainty depth estimation with gated images for 3d reconstruction, 2020.

[31] J. Wang, J. Bartels, W. Whittaker, A. C. Sankaranarayanan, and S. G. Narasimhan. Programmable triangulation light curtains. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[32] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving, 2020.

[33] Z. Xia, P. Sullivan, and A. Chakrabarti. Generating and exploiting probabilistic monocular depth estimates, 2019.

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

[35] G. Yang, P. Hu, and D. Ramanan. Inferring distributions over depth from a single image, 2019.

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

[37] G. J. Yang and B. W. Choi. Smooth trajectory planning along bezier curve for mobile robots with velocity constraints. *International Journal of Control and Automation*, 6(2):225–234, 2013.

[38] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan. Mvsnet: Depth inference for unstructured multi-view stereo, 2018.