Algorithms and Systems for Low Power Time-of-Flight Imaging

by

James Noraky

S.B. in EE, Massachusetts Institute of Technology (2013)
M.Eng. in EECS, Massachusetts Institute of Technology (2014)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 15, 2020

Certified by .....
Vivienne Sze
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Algorithms and Systems for Low Power Time-of-Flight Imaging

by

James Noraky

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Depth sensing is useful for many emerging applications that range from augmented reality to robotic navigation. Time-of-flight (ToF) cameras are appealing depth sensors because they obtain dense depth maps with minimal latency. However, for mobile and embedded devices, ToF cameras, which obtain depth by emitting light and estimating its roundtrip time, can be power-hungry and limit the battery life of the underlying device. To reduce the power for depth sensing, we present algorithms to address two scenarios. For applications where RGB images are concurrently collected, we present algorithms that reduce the usage of the ToF camera and estimate new depth maps without illuminating the scene. We exploit the fact that many applications operate in nearly rigid environments, and our algorithms use the sparse correspondences across the consecutive RGB images to estimate the rigid motion and use it to obtain new depth maps. Our techniques can reduce the usage of the ToF camera by up to 85%, while still estimating new depth maps within 1% of the ground truth for rigid scenes and 1.74% for dynamic ones. When only the data from a ToF camera is used, we propose algorithms that reduce the overall amount of light that the ToF camera emits to obtain accurate depth maps. Our techniques use the rigid motions in the scene, which can be estimated using the infrared images that a ToF camera obtains, to temporally mitigate the impact of noise. We show that our approaches can reduce the amount of emitted light by up to 81% and the mean relative error of the depth maps by up to 64%. Our algorithms are all computationally efficient and can obtain dense depth maps at up to real-time on standard and embedded computing platforms. Compared to applications that just use the ToF camera and incur the cost of higher sensor power and to those that estimate depth entirely using RGB images, which are inaccurate and have high latency, our algorithms enable energy-efficient, accurate, and low latency depth sensing for many emerging applications.

Thesis Supervisor: Vivienne Sze
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

First, I would like to thank my advisor Professor Vivienne Sze for her guidance and support throughout my PhD. She has challenged me to view engineering problems from a practical standpoint and to clearly distill my ideas into clear contributions, which has been an invaluable part of my education. I am also continually amazed by how hard she works and despite that, makes time for all of her students.

I would also like to thank Professor Berthold Horn for providing great technical advice over the past couple of years and of course, for agreeing to be on my committee. I both took and served as a teaching assistant for courses taught by Berthold, which has provided the solid technical footing I needed to apply machine vision techniques to my research. I continue to appreciate and enjoy our technical discussions and am inspired by the great and creative research he does.

I would also like to thank Dr. Charles Mathy for agreeing to be on my committee. Charles has provided both technical advice and friendship over the past years. I met Charles when I was an intern at Analog Devices, where I first got the idea and resources to apply machine vision techniques to depth sensing. Charles is an expert on time-of-flight cameras, and he has provided invaluable insights that made this research possible. He has also made the continued collaboration between MIT and Analog Devices seamless, helping me obtain sensors and various support.

As the last paragraph implies, my collaboration with Analog Devices has been essential for my research. In addition to helpful discussions and inspiring my research direction, the company has generously provided most of the funding for my research. I owe a huge debt of gratitude to Sefa Demirtas, Nicolas Le Dortz, Charles Mathy, Atulya Yellepeddi , and Tao Yu for being my advocates and helping to initiate this collaboration. I also want to thank both Jonathan Yedidia and Zoran Zvonar for their support as well. In addition to these individuals, I would like to thank Kaushal Sanghai for teaching me how to use the company's time-of-flight product and Daniel Schmidt for our helpful discussions on graphics and learning based methods.

I also want to thank my collaborator, Alan Cheng. I mentored Alan as part of

MIT's SuperUROP program and for his M.Eng. With his help, we were able to build an augmented reality demo that showcases the depth estimation work in this thesis.

This PhD journey would not have been possible (or fun) without the support of my colleagues and friends. Over the past 6 years, I am thankful to have wonderful colleagues, who I can also call my friends: Alan, Amr, Diana, Gladynel, Hsin-Yu, Mehul, Nellie, Peter, Soumya, Tien-Ju, Yi-Lun, Yu-Hsin, and Zhengdong. I will always remember our late nights in the lab, group lunches, and our hunts for free food fondly.

I am also fortunate to have a wonderful group of friends outside of lab as well: Albert, Anne, Anuran, Ashwin, Curtis, Chai, David, Eren, Ganesh, Haoyang, Hoon, Isaak, Lehka, Lucas, Mary, Marek, Mohammed, Nirav, Orhan, Tuhin, and Victor. With my friends, I have spent questionable amounts of time sampling the culinary delights of Boston's buffet scene, being cultural patrons of AMC theaters, and having seemingly endless intellectual discussions at the premier, MIT's own Muddy Charles. I am certain that our escapades have delayed my graduation, but I would not take a second of it back. Thank you for making my MIT experience personally rewarding.

I am also thankful to my wonderful and incredibly supportive girlfriend, HanByul. She initiated many of my most memorable adventures in graduate school, some that I hope to repeat (e.g., travel) and others to forget entirely (e.g., spin classes after Thanksgiving day). Thank you for keeping me grounded and sane throughout my entire PhD.

Last and certainly not least, I would like to thank my mom, sister, and grandparents. None of this would have been possible without their love, support, and sacrifice.

Funding: This work was funded by Analog Devices, Inc. Thank you for your generosity and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> Most of the structures in the visual world are rigid or at least nearly so.
>
> *David Marr*

1.1 Motivation

Depth information is useful for many emerging applications, and these range from augmented reality to robotics. Across these different applications, depth information enables realistic and safe interactions with the surrounding environment. For example, in augmented reality, depth information can be used to localize and orient virtual objects as well as detect user gestures [88]. For robotics, depth is important for navigation, enabling tasks like localization and obstacle detection [69]. In order to obtain depth, many of these applications rely on depth sensors. These depth sensors obtain depth in the form of a depth map, which is an image whose pixels represent the distance from the sensor to various points in the scene. An example of a depth map is shown in Figure 1-1.

One appealing sensor that obtains depth maps is a time-of-flight (ToF) camera. These sensors obtain depth by emitting light and estimating its round-trip time. They are appealing because they are compact and obtain dense and accurate depth measurements with minimal latency. All of these features are important for applications,

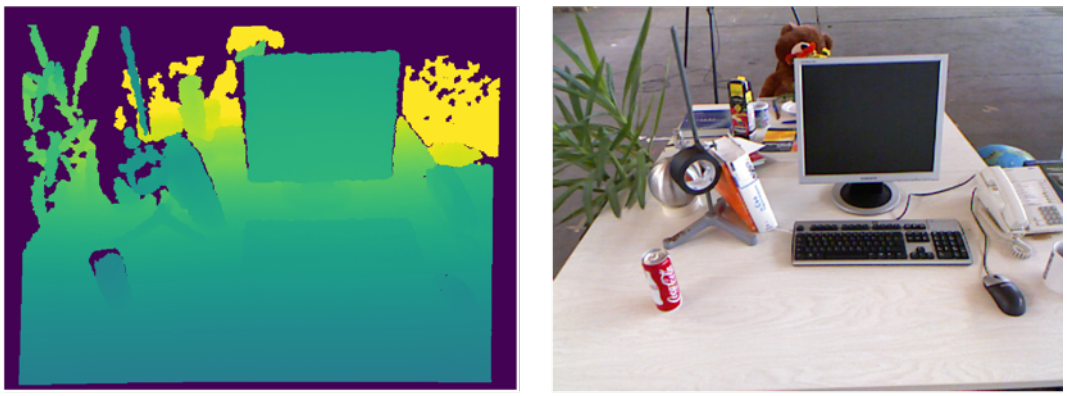


Figure 1-1: **Depth Map Example**: A depth map and its corresponding image is shown. The colors in the depth map represent different depth values.

like robotics, that need depth to react to rapid changes in its environment. However, one potential drawback of a ToF camera is the fact that it must illuminate the scene in order to obtain depth. Depending on the range, these sensors can be power-hungry, especially for applications that need continuous depth measurements. This makes them unappealing for applications that run on mobile and battery-powered devices. Furthermore, the high power consumption of ToF cameras also increases its heat dissipation and forces the addition of cumbersome components such as large heat sinks.

This thesis addresses the high power consumption of ToF cameras by proposing algorithms that reduce the sensor power required to obtain accurate depth maps. To ensure that we can still obtain depth maps with low latency, we are also mindful of the computational complexity of our approaches so that we can obtain dense depth maps in near real-time, or 30 frames per second (FPS), using the low power CPUs of conventional laptop computers and embedded processors. This is a necessary constraint that helps ensure that the power for computation is lower than that of the ToF camera. However, this makes the task of obtaining accurate depth maps challenging because these processors also have limited computational resources. In this thesis, we overcome this challenge by developing algorithms that exploit commonly available data (e.g., RGB images), temporal relationships, and the assumptions made by many applications (e.g., rigidity) to efficiently obtain dense depth maps. In doing so, our

algorithms balance the sensor power of the ToF camera, the accuracy of the resulting depth maps, and the latency which the depth maps are obtained.

In this chapter, we further describe the motivation for our work and provide an overview of the approaches we take. In Section 1.2, we expand on the appeal of ToF cameras by comparing them to other commonly used depth sensors. We show that by emitting light and carefully accumulating its reflection, ToF cameras can reduce the latency required to obtain dense and accurate depth maps. Unfortunately, this comes at the cost of increased power consumption, and in Section 1.3, we corroborate this claim and explain the need for our work. This is followed by an overview of the contributions of this thesis, namely the approaches we take to efficiently estimate accurate and dense depth maps while lowering the sensor power in Section 1.4. Finally, we conclude with an outline of the remainder of this thesis in Section 1.5.

1.2 The Appeal of Time-of-Flight Cameras

As we alluded to in the previous section, there are many sensors that can be used to measure depth. In this section, we briefly describe and compare commonly used sensors that obtain dense depth maps, namely: stereo, active stereo, structured light, and ToF cameras. For each sensor, we comment on its power consumption, the accuracy of its depth maps, and the latency in which they are obtained. As summarized in Table 1.1, we see that a ToF camera offers the best balance between the accuracy of its depth maps and the latency in which they are obtained, making it ideal for applications that need responsive depth in order to react to changes in their environments.

1.2.1 Stereo Camera

Stereo cameras obtain depth for a scene by acquiring and processing two images that are simultaneously captured from two identical cameras that are separated by a distance, or baseline. One advantage of a stereo camera is that it estimates depth in an entirely *passive* fashion, which means that it does not emit light. Because images

Feature	Stereo	Active Stereo	Structured Light	ToF Camera
High Accuracy		✓	✓	✓
Low Latency				✓
Low Power	✓			

Table 1.1: **Depth Sensor Comparison**: We compare stereo, active stereo, structured light, and ToF cameras in terms of their accuracy, latency, and power consumption.

can be acquired in an energy-efficient manner, stereo cameras estimate depth with low power.

To estimate depth from these two images, stereo algorithms use the pixel-wise displacements between the captured images. Because these images are taken from different positions, the pixels that correspond to the same objects are displaced from each other. Furthermore, these displacements are a function of the distance between the object and the stereo camera, where the pixels of an object further away have smaller displacements than those for objects at a closer distance. Stereo cameras and their underlying algorithms exploit this property to obtain a depth map by first finding the corresponding pixels across the two images and then using this information to obtain depth [28]. This process in shown in Figure 1-2, where we assume that the two camera images are rectified. From this figure, we see that we can obtain depth by exploiting the relationship between similar triangles (shown in red and green), where:

$$\frac{B}{Z_i} = \frac{B - x_i + x_i'}{Z_i - f} \tag{1.1}$$

where B is length of the baseline that separates the two cameras, f is the focal length, x_i and x_i' are the x-coordinate of the i^{th} pixel and its correspondence, and Z_i is its depth. By rearranging the terms of Eq. (1.1), we obtain the following expression for Z_i:

$$Z_i = \frac{Bf}{x_i - x_i'} \tag{1.2}$$

In order to determine the corresponding pixels between the two images, block matching or optical flow algorithms are often used [79]. These algorithms assume

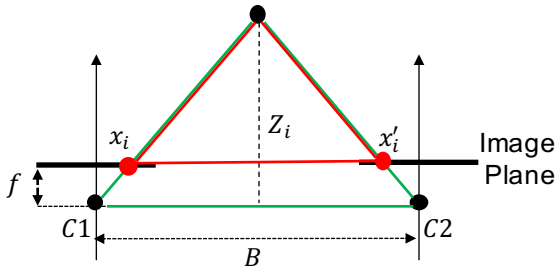


Figure 1-2: **Stereo Camera**: We show the configuration of a stereo camera, and the relevant variables used to estimate depth for the i^{th} pixel (Z_i): the two camera centers ($C1$ and $C2$), their focal length (f), the baseline distance that separates the cameras (B), and the x-coordinate of the i^{th} pixel and its correspondence (x_i and x_i').

that corresponding pixels and their neighboring regions have similar intensities. As such, block matching algorithms take a block of pixels from one image and compare it to candidates in the other image. Optical flow algorithm relate the pixel-wise displacements to the gradients of the images to estimate the per-pixel displacements iteratively [37]. To obtain a dense depth map, stereo algorithms need dense correspondences, which are computationally expensive to compute [19]. For a laptop computer and an embedded processor, this reduces the frame rate and increases the latency in which depth maps are obtained.

Furthermore, it is also not always *possible* to accurately determine corresponding pixels, which negatively impacts the accuracy of the estimated depth maps. For example, in untextured regions, finding correspondences is inherently ambiguous. This is also true along edges due to the aperture effect [34]. Due to these ambiguities, it is not possible to estimate dense depth maps for every scene. To address this challenge, many stereo algorithms enforce constraints like spatial smoothness to infill these ambiguous regions. This comes with an additional computational cost, and these heuristics fail when the assumptions are violated [79].

Another way to overcome this issue is to apply a texture directly onto the object or scene, which is the approach taken by active stereo and structured light cameras.

1.2.2 Active Stereo Camera

Active stereo cameras, like their passive counterparts, estimate depth using images captured by two cameras separated by a baseline. As implied by its name, these sensors are *active* sensors, which means that they emit light in order to estimate depth. In particular, active stereo cameras use an infrared projector to project a texture onto the environment so that corresponding pixels can be determined regardless of the actual texture in the scene [41]. Being infrared, the projected texture is invisible to humans, but is visible to the infrared cameras and algorithms used to determine the correspondences. Therefore, active stereo cameras estimate accurate and dense depth maps for the region that corresponds to the overlap between the field-of-views of the infrared cameras and the projector. While the accuracy of this sensor is limited by the resolution of the projector, it nonetheless addresses a major challenge faced by stereo cameras.

However, this comes at a cost of increased power consumption, especially for longer ranges. Furthermore, this sensor still needs to find dense correspondences for the pixels in the overlapping field-of-views, and thus, the high computational costs and high latency remain.

1.2.3 Structured Light Camera

Structured light cameras are similar to active stereo cameras in that they also project a coded pattern onto the scene, but differ in that they estimate depth based on how the pattern, captured by a *single* infrared camera, deforms. This deformation, or the correspondence between the pixels in the captured image and those of the projected pattern, is typically obtained by using stereo matching algorithms. Examples of structured light cameras include the first generation Kinect and the front facing depth sensor in the iPhone [6]. One advantage of this sensor over an active stereo camera is that it reduces the number of cameras required to estimate depth. However, the high computational costs of determining the deformations for a *dense* depth map remain. This sensor is also power-hungry for longer ranges, and the accuracy of its depth maps

is also limited by the resolution of its projector.

All of the sensors described so far estimate depth using dense correspondences (or deformations in the case of a structured light camera), which is challenging to accurately and efficiently compute. In contrast, ToF cameras estimate depth using a different approach that allows it to obtain dense depth measurements with minimal computation and latency.

1.2.4 Time-of-Flight Camera

As we previously described, ToF cameras obtain depth by emitting infrared light and estimating its round-trip travel time. This time-of-flight principle is also used by sensors like LIDAR, which measures depth by physically scanning the scene with laser light. What differentiates a ToF camera is that it uses diffuse light in order to obtain dense depth measurements. As a result, ToF cameras have no moving parts and are compact, making them ideal to integrate into larger systems. This is another advantage over the stereo and structured light cameras, which require large baselines to estimate depth further away. In this section, we focus on ToF cameras, which are grouped into pulsed and continuous wave cameras.

Pulsed Time-of-Fight Camera

Pulsed ToF cameras obtain depth by emitting pulses of light and estimating its round-trip travel time. This round-trip time can be estimated for each pixel by accumulating the reflected light over a sensor array. This process is shown in Figure 1-3 for a single emitted pulse and pixel, where τ is the pulse width or the duration in which light is emitted. We see that its reflection is accumulated over different time intervals. We denote $S0$ as the reflected light that is accumulated over the same time interval in which the pulse is being emitted. We also accumulate light for an equal duration afterwards and denote this as $S1$. As shown in the figure, the quantity $S0 + S1$ is proportional to the pulse width, where its area is a function of the intensity of the emitted light and the material properties of the reflected object. Furthermore, $S1$

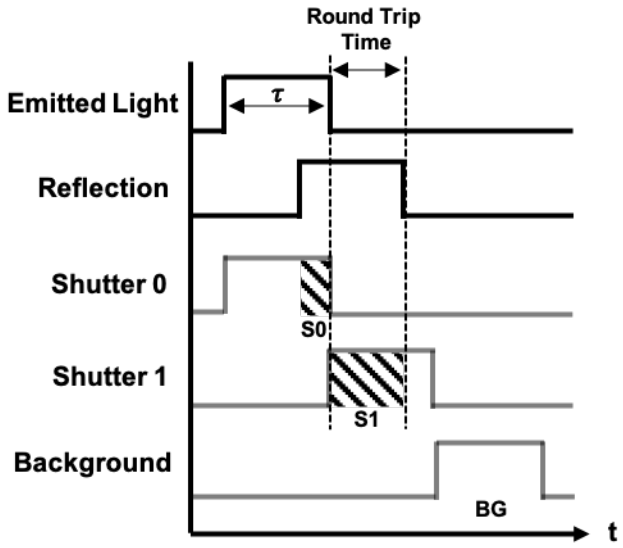


Figure 1-3: **Pulsed ToF Camera**: We depict the operation of a pulsed ToF camera for a single pulse and pixel. The reflected light is accumulated over two intervals with an additional period used to account for the background light, from which the round-trip time and depth can be obtained. In practice, multiple pulses are emitted and accumulated to reduce noise.

is proportional to the round-trip travel time of the light. Assuming that there is no contribution from the background, we see that the depth for the i^{th} pixel located at (x_i, y_i), or Z_i, is given by:

$$Z_i = \left(\frac{c\tau f}{2\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2 + f^2}} \right) \frac{S1_i}{S1_i + S0_i} \tag{1.3}$$

where c is the speed of light, f is the focal length of the ToF camera, and (x_c, y_c) is its principal point. In practice, there is background infrared light that is also accumulated in $S0_i$ and $S1_i$ and thus, there is an additional shutter period to estimate this background contribution. We denote this as BG in Figure 1-3. With this measurement, we can obtain the corrected depth as follows:

$$Z_i = \left(\frac{c\tau f}{2\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2 + f^2}} \right) \frac{S1_i - BG_i}{S1_i + S0_i - 2BG_i} \tag{1.4}$$

Furthermore, in addition to obtaining depth maps, the accumulated light can also be used to obtain an infrared image as shown in Figure 1-4.

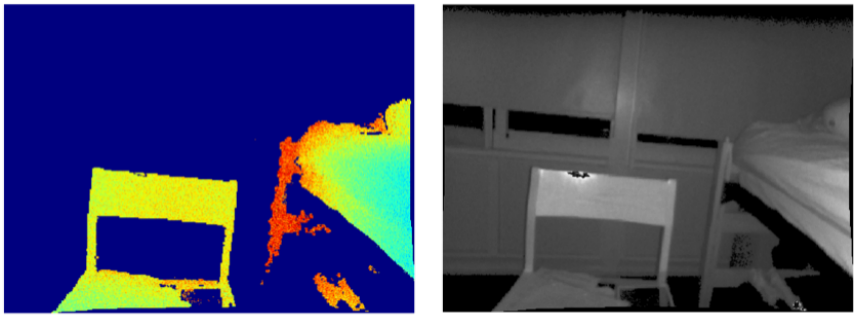


Figure 1-4: **Data from ToF Camera**: In the processing of obtaining a depth map, ToF cameras also obtain an infrared image.

Both Eq. (1.3) and Eq. (1.4) assume that Z_i is at most $\frac{c\tau}{2}$ meters, and τ can be varied depending on the application. Therefore, the low computation and latency of these sensors is clear given that depth can be estimated using simple arithmetic operations as opposed to finding dense and ambiguous correspondences.

Continuous Wave Time-of-Flight Cameras

Another variant of a ToF camera is known as a continuous wave (CW) ToF camera. As shown in Figure 1-5, CW ToF cameras obtain depth by emitting sinusoidal-modulated light and estimating the *phase difference* between the emitted light and its reflection. To estimate the phase difference, a common approach is to compute the cross-correlation between the signals that represent the emitted and reflected light. We denote the emitted light as:

$$s(t) = A_s \cos(\omega t) + B_s \tag{1.5}$$

where A_s and B_s are the magnitude and offset of the emitted light, respectively, and ω is the angular frequency of the modulated light. We denote its reflection as:

$$r(t) = A_r \cos(\omega t - \phi) + B_r \tag{1.6}$$

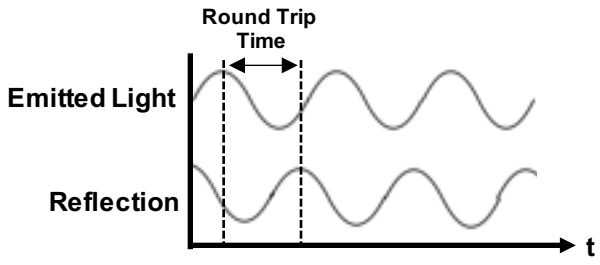


Figure 1-5: **Continuous Wave ToF Camera**: Unlike a pulsed ToF camera, a CW ToF camera emits sinusoidal-modulated light and estimates the phase difference between the emitted light and its reflection to obtain depth.

where A_r and B_r are the magnitude and offset of the reflected light, respectively, and ϕ is the phase difference. The cross-correlation, $c(t)$, is given by:

$$c(t) = \frac{A_r A_s}{2} \cos(\omega t + \phi) + B_r B_s \tag{1.7}$$

To estimate ϕ, we sample Eq. (1.7) at 4 equally spaced samples with $t = \{0, \frac{\pi}{2\omega}, \frac{\pi}{\omega}, \frac{3\pi}{2\omega}\}$ to obtain $S0$, $S1$, $S2$, and $S3$, respectively. With these quantities, the phase can be obtained as follows:

$$\phi = \tan^{-1}\left(\frac{S3 - S1}{S0 - S2}\right) \tag{1.8}$$

This process can be performed efficiently at each pixel by accumulating the reflected light, and the depth of the i^{th} pixel located at (x_i, y_i) is given by:

$$Z_i = \left(\frac{f}{\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2 + f^2}}\right) \frac{c\phi_i}{2\omega} \tag{1.9}$$

where ϕ_i is the phase offset of the i^{th} pixel.

One benefit of this approach is that there is no need to measure the background infrared light as the difference in the numerator and denominator of Eq. (1.8) accounts for this. Because the phase wraps around every 2π, we see that depth can only be measured unambiguously up to $\frac{c\pi}{\omega}$ meters. For high modulation frequencies, this distance is small, and many CW ToF cameras use two coprime modulation frequencies and the Chinese Remainder Theorem to resolve this ambiguity and extend the range

of the ToF camera [25].

Regardless of the underlying approach used, ToF cameras estimate depth efficiently compared to stereo, active stereo, and structured light cameras. By carefully acquiring the reflected light, ToF cameras estimate depth using simple, pixel-wise operations instead of computationally expensive correspondence searches. However, this advantage brought about by illuminating the scene can be a drawback for battery-powered devices, which we discuss in the next section.

1.3 Time-of-Flight Camera Power Consumption

Because ToF cameras must illuminate a scene in order to measure depth, these sensors can be power-hungry, and its power consumption depends on the range of the sensor. As shown in Table 1.2, commercial ToF cameras consume anywhere between 300 mW and 20 W. If an application only requires short range depth intermittently, then its power consumption may be negligible. However, this is not true for many of the applications we consider, which need depth continuously. For example, many augmented reality applications need continuous depth in order to reconstruct the local environment to allow users to interact with it. Furthermore, long range depth is essential for robotic applications in order to navigate and quickly react to obstacles in its surroundings. For both of these scenarios, it is critical to lower the power for depth sensing to extend the battery life of the underlying device as well as reduce the dissipated heat.

Model	**Range (m)**	**Power (W)**
Basler [3]	0 - 13	15
IFM O3D302 [39]	0.3 - 8	10
Pico Flexx [70]	0.1 - 4	0.3
Pico Monstar [71]	0.5 - 6	4.5
Swift-E [66]	0.5 - 6	20

Table 1.2: **ToF Camera Power Consumption**: The range and power consumption according to the specifications of different ToF cameras is shown. It should be noted that these ranges are the maximum possible under *ideal* scenarios.

To reduce the power for depth sensing, it is appealing to simply reduce the total amount of light that the ToF camera emits and use the resulting depth map. However, doing this alone not only reduces the range of the ToF camera, but also increases the noise in the depth estimates. This is because the accumulated light is also affected by noise, and it is pronounced when the intensity of the reflected light is low. In Chapter 5, we provide a more rigorous analysis of this noise, but we can visualize its impact in Figure 1-6. We see that one immediate consequence of lowering the power is not just reduced range but also reduced depth resolution, where features of the bust cannot be resolved.

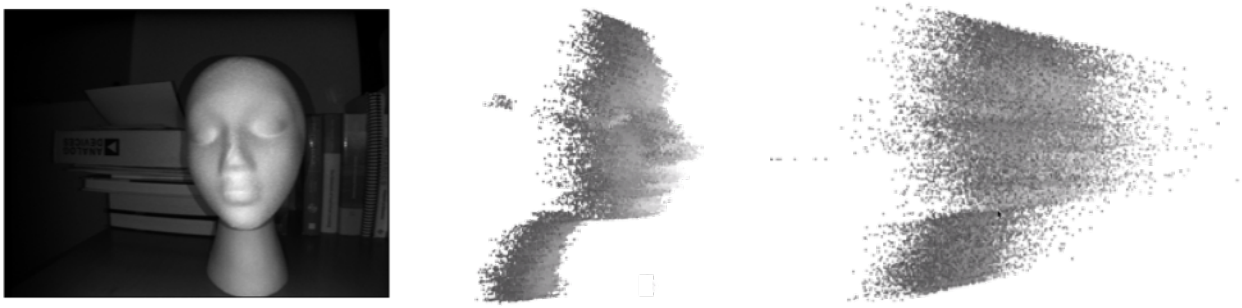


Figure 1-6: **Impact of Power**: We show the impact of reducing the amount of emitted light by visualizing the resulting point clouds. These depth maps were captured using an ADI ToF Camera [13].

In this thesis, we propose algorithms that lower the power for depth sensing while still obtaining accurate and dense depth maps. To ensure that the depth maps can be obtained with minimal latency, we ensure that our techniques are computationally efficient so that we can obtain depth maps at up to real-time (30 FPS) on the CPUs of a standard laptop computer and embedded processor. We provide an overview of these techniques in the next section.

1.4 Thesis Contributions

To lower the sensor power of ToF cameras, we propose algorithms that can be broadly categorized into the following strategies:

- **Depth Map Estimation Using RGB Images**: When RGB images are concurrently collected, we reduce the duty cycle of the ToF camera and use consecutive RGB images and a previous depth map to estimate new depth maps without turning on the ToF camera.
- **Low Power Temporal Depth Filtering**: When only the ToF camera is used, we reduce the overall amount of emitted light and use the IR images that a ToF camera collects to mitigate the effects of noise.

Both of these strategies use the temporal relationship, namely the 3D scene motion, across consecutive frames to lower the sensor power of the ToF camera while obtaining low latency and accurate depth maps. Our algorithms estimate the 3D motion for the following scenarios:

- **Rigid Objects**: The motion of these objects can be described using a single rotation and translation, which preserves the pairwise distance between any two points.
- **Non-Rigid Objects**: These objects deform such that the pairwise distance between any two points may change, but otherwise remain intact. An example of a non-rigid object is a bending sheet of paper.
- **Dynamic Scenes**: These scenes contain multiple objects undergoing independent, rigid and non-rigid motions.

These scenarios encompass a wide variety of scenes and applications.

1.4.1 Depth Map Estimation Using RGB Images

One simple way to reduce the sensor power of a ToF camera is to reduce its usage as shown in Figure 1-7. However, this is insufficient for applications that need depth

maps in real-time, or 30 FPS. To maintain the rate at which depth maps are acquired, we propose algorithms that leverage concurrently collected and consecutive RGB images and a previous depth map to estimate a new depth map as shown in Figure 1-8. In many applications, RGB images are typically collected for other purposes, and we reuse them to estimate depth. Our algorithms assume that the pixels of the RGB images and depth maps are spatially aligned and use the pixel-wise motion of these images to estimate the 3D scene motion and new depth maps sequentially.

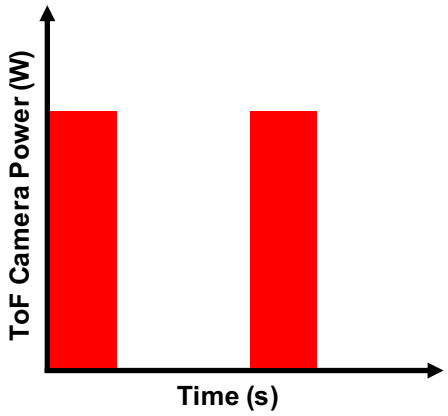


Figure 1-7: **Reduce the Usage of ToF Camera**: By reducing the usage of the ToF camera, we reduce the overall sensor power.

For the different objects and scenes that we consider, we show how the assumption of rigidity can be used to efficiently estimate accurate depth maps. In particular, our contributions are:

- When the scene contains a rigid object, we show that its 3D motion can be estimated using the optical flow at a *sparse set of its pixels* with *linear least squares*. This is relevant for many applications, which range from augmented reality to robotic navigation, that operate in static environments. By modeling these scenes as a rigid object, our algorithm is able to estimate dense and accurate depth maps in real-time on low power embedded processors and lower the overall system power for depth sensing (Chapter 3).

- For non-rigid objects and dynamic scenes, we show that we can still leverage the assumption of rigidity. By modeling these objects and scenes as *locally*

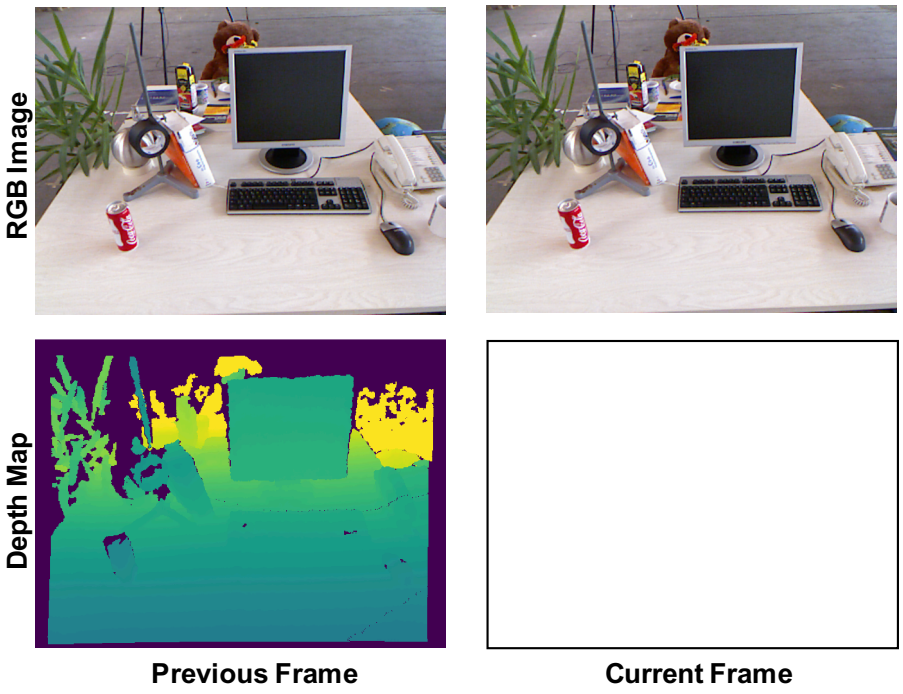


Figure 1-8: **Depth Map Estimation**: Our algorithms estimate new depth maps using concurrently collected and consecutive RGB images without using the ToF camera.

rigid, our algorithms use the previous depth map and the optical flow at a sparse set of pixels to estimate the rigid motions in the scene and assign them without computationally expensive operations like rigid motion segmentation. As a result, our algorithms can estimate accurate depth maps in up to real-time using the CPUs of a standard laptop computer and increase the variety of applications that our framework can support (Chapter 4).

The resulting algorithms can reduce the usage of the ToF camera by up to 85% while obtaining depth maps within 1% of what the ToF camera measures for rigid scenes and within 1.74% for non-rigid and dynamic ones. Additionally, we also show that our estimated depth maps can be used in a real augmented reality application in Section 3.5.

1.4.2 Low Power Temporal Depth Filtering

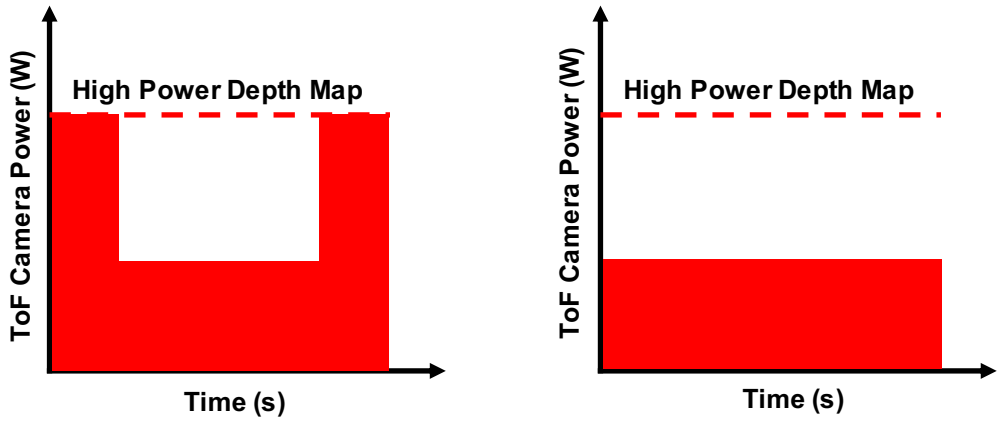
We also consider the case where RGB images are not concurrently collected and only use the data obtained by a ToF camera to reduce the power required to obtain accurate depth maps. This is important for applications with small form factors that lack the additional space for an image sensor. Instead of obtaining high power depth maps for each frame, our algorithms reduce the light the ToF camera emits and use the consecutive infrared images that a ToF camera collects in the process of obtaining a depth map to mitigate the effects of noise. One of the challenges of using multiple sensors is to synchronize the different images and depth maps, but in this scenario, this is not required as the depth map and infrared images are from the same source. In contrast to the scenario in Section 1.4.1, the ToF camera is *always on*, and our goal is to reduce the overall amount of emitted light while still obtaining accurate depth maps.

To mitigate the effects of noise, we use the infrared images to estimate the rigid motions in the scene. Regardless of the noise in the underlying infrared images, we find that they are accurate enough to estimate *sparse optical flow*, which we use to obtain the rigid motions. This allows us to temporally combine the depth maps across consecutive frames efficiently, enabling the following schemes:

- When the amount of light that a ToF camera emits can be varied, we propose an algorithm that intermittently obtains high power depth maps and uses the estimated rigid motions to combine them with subsequent low power depth maps to denoise the low power depth maps. As shown in Figure 1-9a, this approach reduces the overall amount of power required to obtain accurate depth maps (Chapter 5).

- When only low power depth maps can be acquired (e.g., for sensors with limited illumination sources) as shown in Figure 1-9b, we propose an algorithm that denoises these depth maps by recursively filtering them. Our algorithm estimates the rigid motion between the consecutive depth maps and uses the rigid motion to efficiently register them as well as account for the changes in

depth to mitigate the effects of noise (Chapter 5).



(a) Adaptively Vary the Emitted Light (b) Temporally Filter Depth Maps

Figure 1-9: **Reduce the Amount of Emitted Light**: We show the different scenarios in which we reduce the overall amount of light that a ToF camera emits. In order to mitigate the effect of noise, our algorithms combine the consecutive depth maps by estimating the rigid motion using the infrared images.

In general, temporal filtering techniques are computationally expensive due to the need of processing all of the pixels across many frames. By using the assumption of rigidity, we are able to efficiently obtain accurate depth maps at up to real-time using the CPUs of an embedded processor, even when the amount of emitted light is reduced by over 80%.

1.4.3 Evaluation Framework

All of our algorithms balance the sensor power of the ToF camera, the accuracy of the resulting depth maps, and the latency in which the depth maps are estimated to obtain a favorable tradeoff. In this thesis, we quantify this using the following metrics:

Duty Cycle or Power Reduction (%) To quantify the sensor power of the ToF camera, we use either the duty cycle or power reduction. The *duty cycle* is the percentage of depth maps acquired using the ToF camera. This metric is used to evaluate our approaches that reduce the usage of the ToF camera and estimate

new depth maps using RGB images instead. The *power reduction* is the percentage reduction in the power used to obtain depth maps compared to its regular operation, where high power depth maps are obtained for each frame. We use this metric to evaluate our adaptive power depth map scheme as shown in Figure 1-9a, where the ToF camera is always on but emits less light overall.

Mean Relative Error (%) To quantify the accuracy of the depth maps that our algorithms obtain, we use the *mean relative error*. This metric is equal to:

$$\text{MRE} = \frac{100}{N} \sum_{i=1}^{N} \frac{|Z_i - \hat{Z}_i|}{Z_i} \tag{1.10}$$

where Z_i is the ground truth depth for the i^{th} pixel, $\hat{Z}_i$ is the estimate, and N is the total number of pixels. This metric penalizes a unit error more so up close than further away, which is ideal for applications like robotic navigation that need to quickly react to changes in its immediate environment. For our evaluations, this metric also allows us to compare the performance of our algorithm on datasets with different dynamic ranges.

Estimation Frame Rate (FPS) To quantify the latency in which our algorithms obtain dense depth maps, we measure its frame rate on the ODROID-XU3 embedded processor (Cortex-A7/Cortex-A15 Octa Core CPU) [26] and a 2015 MacBook Pro (i5-5257U Dual Core CPU) [15]. We use these platforms because they are ubiquitous (e.g., the CPU of the ODROID-XU3 board is used in the Samsung Galaxy S5 [78]) and to ensure that the power for computation is lower than that of a ToF camera.

With these metrics, we show that compared to approaches that rely heavily on computation and those that strictly use the ToF camera in a high power setting, our algorithms enable energy-efficient, accurate, and low latency depth sensing.

1.5 Thesis Outline

This thesis is organized as follows. In Chapter 2, we describe how we estimate the rigid motion using consecutive images and a previous depth map. This is the foundation for the different approaches proposed in this thesis. In Chapter 3, we describe how we can use concurrently collected RGB images to reduce the usage of the ToF camera and estimate depth maps for a single rigid object. This is useful for applications that interact in static environments. In Chapter 4, we show how RGB images can be used to estimate depth maps for non-rigid objects and dynamic scenes. This enables our framework to support a larger variety of applications. We address the case when RGB images are not available in Chapter 5. We show how we use the infrared images to estimate both rigid and dynamic motions, which allows us to reduce the amount of light the ToF camera emits and denoise the low power depth maps. We conclude in Chapter 6, where we summarize our key insights and discuss future work.

1.5.1 Summary of Publications

Here, we list the publications that went into this thesis. In Table 1.3, we summarize the scope for each of our publications.

No.	Year	Venue	Section	RGB	Rigid	Non-Rigid	Dynamic
1	2017	ICIP	3.1	✓	✓		
2	2018	ICIP	4.3	✓		✓	
3	2019	ICIP	5.3				✓
4	2020	TCSVT	3.1	✓	✓		
5	2020	arXiv	4.5	✓			✓
6	2020	-	5.6		✓		

Table 1.3: **Publications**: We categorize each of the publications that went into this thesis. If the approach uses RBG images, we check the **RGB** column. We also indicate whether the publication is focused on rigid objects (**Rigid**), non-rigid objects (**Non-Rigid**), or dynamic scenes (**Dynamic**).

Conference Publications

1. J. Noraky and V. Sze, "Low Power Depth Estimation for Time-of-Flight imaging," IEEE International Conference on Image Processing (ICIP), Beijing, China, 2017, pp. 2114-2118.

2. J. Noraky and V. Sze, "Depth Estimation of Non-Rigid Objects for Time-of-Flight Imaging," IEEE International Conference on Image Processing (ICIP), Athens, Greece, 2018, pp. 2925-2929.

3. J. Noraky, C. Mathy, A. Cheng and V. Sze, "Low Power Adaptive Time-of-Flight Imaging for Multiple Rigid Objects," IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 2019, pp. 3517-3521.

Journal Publications and Preprints

4. J. Noraky, V. Sze, "Low Power Depth Estimation of Rigid Objects for Time-of-Flight Imaging," IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), 2020.

5. J. Noraky, V. Sze, "Depth Map Estimation of Dynamic Scenes Using Prior Depth Information," Under Review, February 2020. Available: `http://arxiv.org/abs/2002.00297`

6. J. Noraky, V. Sze, "Low Power Depth Map Denoising for Mobile Time-of-Flight Cameras," In Preparation, 2020.

Chapter 2

Rigid Motion Estimation

As summarized in Chapter 1, the algorithms in this thesis lower the sensor power of time-of-flight (ToF) cameras by estimating the rigid motions in the scene. In this chapter, we review the common assumptions and approaches used by our different techniques to estimate this 3D motion using the data from consecutive frames.

In Section 2.1, we first introduce the image formation model that we use to describe how 3D objects and their motions are mapped onto 2D images. Our techniques invert this model by using the 2D pixel-wise motion of the consecutive images to estimate the 3D scene motion. When this motion is rigid, we show in Section 2.2 that it can be estimated using the pixel-wise motion at a *sparse* set of pixels. As we demonstrate throughout this thesis, this allows us to lower the latency in which we estimate accurate depth maps for a variety of different scenarios. Finally, in Section 2.3, we summarize the notation that we introduced, which will be used for the remainder of this thesis.

2.1 Image Formation Model

Our approaches assume perspective projection, which describes how 3D objects are projected onto 2D images. As shown in Figure 2-1, the 2D projection of the 3D point, $\boldsymbol{P_i} = (X_i, Y_i, Z_i)^T$, is found by taking the intersection of the line that connects $\boldsymbol{P_i}$ to the *center of projection* with the *image plane*. Denoting the 2D coordinate of the

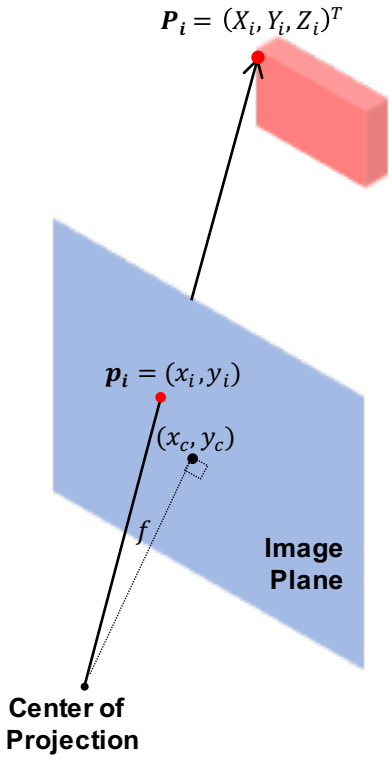


Figure 2-1: **Perspective Projection**: We assume images are formed by perspective projection as described by Eq. (2.1).

projection as $\boldsymbol{p_i} = (x_i, y_i)^T$, we have the following relationship:

$$\frac{x_i - x_c}{f} = \frac{X_i}{Z_i} \quad \text{and} \quad \frac{y_i - y_c}{f} = \frac{Y_i}{Z_i} \tag{2.1}$$

where f is the principal distance (focal length), and (x_c, y_c) is the principal point (center). As shown in the figure, the principal point is the perpendicular intersection of the line that connects the center of projection to the image plane, and the length of that segment is the principal distance.

In our algorithms, we use the relationship in Eq. (2.1) to determine the 3D position of each pixel in a depth map. For the i^{th} pixel, we have:

$$\boldsymbol{P_i} = \frac{Z_i}{f}(x_i - x_c, y_i - y_c, f)^T \tag{2.2}$$

where Z_i is its depth taken from the depth map. Conversely, given the 3D coordinate

of a point, we can also obtain the 2D coordinate of its projection. This is the basis for the *reprojection* operation that we use in our algorithms to obtain new depth maps. From Eq. (2.1), we have:

$$x_i = f\frac{\hat{\boldsymbol{x}} \cdot \boldsymbol{P_i}}{\hat{\boldsymbol{z}} \cdot \boldsymbol{P_i}} + x_c \quad \text{and} \quad y_i = f\frac{\hat{\boldsymbol{y}} \cdot \boldsymbol{P_i}}{\hat{\boldsymbol{z}} \cdot \boldsymbol{P_i}} + y_c \tag{2.3}$$

where $\cdot$ denotes the dot product and $\hat{\mathbf{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}$ are the unit vectors oriented along the coordinate axes.

Naturally, as the 3D point moves, its 2D pixel location also moves. We denote the 3D displacement of $\boldsymbol{P_i}$ as $\dot{\boldsymbol{P}}_{\boldsymbol{i}} = (\dot{X}_i, \dot{Y}_i, \dot{Z}_i)^T$, where we use the $\dot{}$ notation to indicate its temporal displacement. Therefore, its new 3D position, $\boldsymbol{P'_i}$, is given by:

$$\boldsymbol{P'_i} = \boldsymbol{P_i} + \dot{\boldsymbol{P}}_{\boldsymbol{i}} \tag{2.4}$$

where we use $'$ to denote its new *corresponding* location. Using Eq. (2.3), we can then obtain its new pixel location, $\boldsymbol{p'_i} = (x'_i, y'_i)$, as follows:

$$x'_i = f\frac{X_i + \dot{X}_i}{Z_i + \dot{Z}_i} + x_c \tag{2.5}$$

$$y'_i = f\frac{Y_i + \dot{X}_i}{Z_i + \dot{Z}_i} + y_c \tag{2.6}$$

This is the relationship that our algorithms exploit in order to estimate the 3D scene motion using the data in our problem setup as shown in Figure 2-2. To solve this system, one common approach is to first linearize these rational equations and then solve the resulting linear system. However, we still cannot solve this linear system because it is *underdetermined*. Our algorithms overcome this by assuming *rigid motion*, which not only converts this underdetermined system into an overdetermined one, but also allows us to estimate the 3D motion of all the pixels of a rigid object using the data from a *sparse* subset of its pixels.

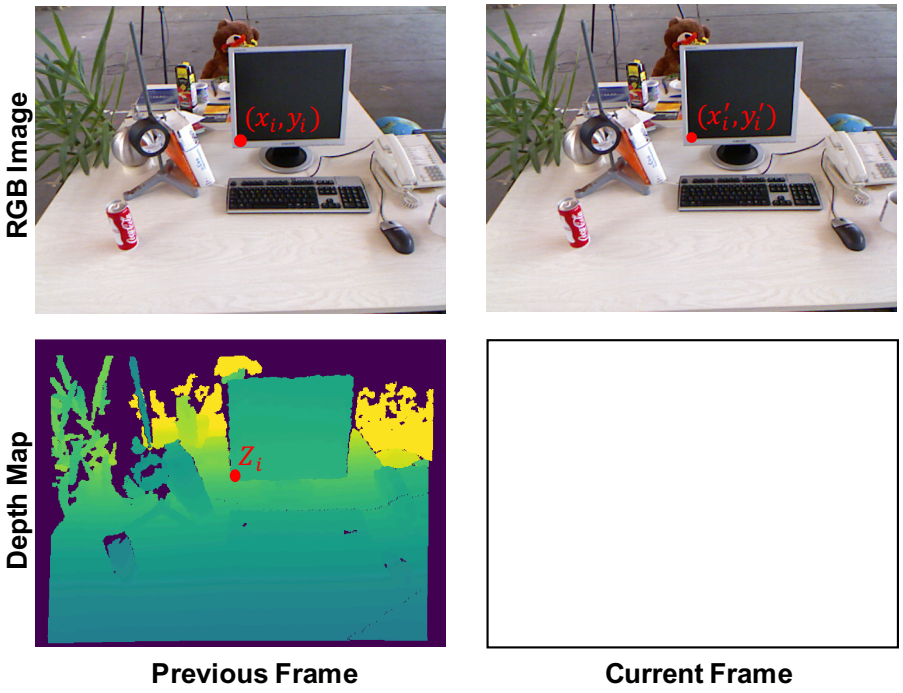


Figure 2-2: **Problem Setup**: Our algorithm estimates the 3D scene motion between the consecutive frames by using the depth and 2D motion of the pixels in the previous frame. The depth of the pixel located at (x_i, y_i) in the previous frame is given by Z_i. We can use this information to obtain its 3D position, or $\boldsymbol{P_i}$, using Eq. (2.3). Due to the motion in the scene, $\boldsymbol{P_i}$ moves to $\boldsymbol{P_i'}$, and its new pixel location is given by (x_i', y_i').

2.2 Rigid Motion Estimation

If $\boldsymbol{P_i}$ undergoes rigid motion, we can obtain its new corresponding position, $\boldsymbol{P_i'}$, using its *pose*. The pose is composed a 3×3 rotation matrix, $\boldsymbol{R}$, and a 3D vector, $\boldsymbol{T}$. It allows us to rewrite Eq. (2.4), which describes $\boldsymbol{P_i'}$ in terms of a *general motion*, in terms of its rigid motion:

$$\boldsymbol{P_i'} = \boldsymbol{R}\boldsymbol{P_i} + \boldsymbol{T} \tag{2.7}$$

Many methods estimate the pose by using the 2D coordinates of $\boldsymbol{P_i}$ and $\boldsymbol{P_i'}$ to first estimate the essential matrix and then factor it to obtain $\boldsymbol{R}$ and $\boldsymbol{T}$. The essential matrix can be estimated using techniques that range from performing a singular value decomposition (when 8 corresponding pixels are known) [54] to finding the roots of a tenth order polynomial (when 5 corresponding pixels are known) [60]. The essential matrix can then be factored to obtain 4 potential poses [28, 33].

We do not use these methods because they do not utilize the depth data and other assumptions of our problem setup. By taking these into account, we show that the rigid motion can be directly estimated with *fewer correspondences* using *linear least squares*. As shown in Chapters 3 to 4, this is advantageous when used with techniques like RANSAC [18], especially when there is noise in the depth and pixel-wise motion data.

2.2.1 Instantaneous Approximation

When the 3D motion between consecutive frames is small, which is a reasonable assumption for applications where frames are captured at 30 FPS or higher, we can linearize Eq. (2.5) and Eq. (2.6) by using their Taylor approximation:

$$x_i' = f\frac{X_i + \dot{X}_i}{Z_i + \dot{Z}_i} + x_c \approx f\frac{X_i}{Z_i} + f\frac{\dot{X}_i}{Z_i} - f\frac{X_i}{Z_i^2}\dot{Z}_i + x_c \tag{2.8}$$

$$y_i' = f\frac{Y_i + \dot{Y}_i}{Z_i + \dot{Z}_i} + y_c \approx f\frac{Y_i}{Z_i} + f\frac{\dot{Y}_i}{Z_i} - f\frac{Y_i}{Z_i^2}\dot{Z}_i + y_c \tag{2.9}$$

By using Eq. (2.3), we can simplify Eq. (2.8) and Eq. (2.9) to obtain:

$$\dot{x_i} = \frac{f\dot{X}_i - x_i\dot{Z}_i}{Z_i} \tag{2.10}$$

$$\dot{y_i} = \frac{f\dot{Y}_i - y_i\dot{Z}_i}{Z_i} \tag{2.11}$$

where $\dot{x_i} = x_i' - x_i$ and $\dot{y_i} = y_i' - y_i$ are the pixel-wise motion of the i^{th} pixel. In our problem setup (Figure 2-2), the depth in the previous frame (Z_i) is known, and thus, Eq. (2.10) and Eq. (2.11) *linearly* relate the 3D motion of the i^{th} point in the previous frame, $\dot{X}_i, \dot{Y}_i, \dot{Z}_i$, to its 2D pixel-wise motion, $\dot{x_i}, \dot{y_i}$.

To obtain an expression for $\dot{x_i}$ and $\dot{y_i}$ in terms of the rigid motion, we need to rewrite $\dot{X}_i, \dot{Y}_i, \dot{Z}_i$ in terms of the parameters of the pose. However, one challenge is that the rotation matrix, $\boldsymbol{R}$, has non-linear constraints. This would prevent the rigid motion parameters from being estimated using linear least squares. To overcome this challenge, we again make use of the fact that the time between frames is small, which allows us to represent $\dot{X}_i, \dot{Y}_i, \dot{Z}_i$ using its *instantaneous rotation*. We will show that the resulting formulation is linear.

To do so, we first define the quantity $\boldsymbol{P_i(t)}$, which is the 3D position of the i^{th} point at time t, as:

$$\boldsymbol{P_i(t)} = \boldsymbol{R(t)P_i(0)} + \boldsymbol{D(t)} \tag{2.12}$$

where $\boldsymbol{P_i(0)}$ is the initial position of the point, $\boldsymbol{R(t)}$ is a 3×3 rotation matrix that describes its orientation, and $\boldsymbol{D(t)}$ is a 3D vector that describes its displacement from the origin. Both $\boldsymbol{R(t)}$ and $\boldsymbol{D(t)}$ vary with time. To obtain $\dot{X}_i, \dot{Y}_i, \dot{Z}_i$, we take the derivative of Eq. (2.12) with respect to time. We have:

$$\boldsymbol{\dot{P}_i(t)} = \boldsymbol{\dot{R}(t)P_i(0)} + \boldsymbol{\dot{D}(t)} = \boldsymbol{\dot{R}(t)R(t)^T R(t)P_i(0)} + \boldsymbol{\dot{D}(t)} \tag{2.13}$$

where $\boldsymbol{\dot{P}_i(t)} = (\dot{X}_i, \dot{Y}_i, \dot{Z}_i)^T$ is the derivative of $\boldsymbol{P_i(t)}$, $\boldsymbol{\dot{R}(t)}$ is the derivative of $\boldsymbol{R(t)}$, and $\boldsymbol{\dot{D}(t)}$ is the derivative of $\boldsymbol{D(t)}$. The second equality in Eq. (2.13) is obtained by

using the identity $\boldsymbol{R(t)^T R(t) = I}$. This allows us to rewrite Eq. (2.13) as:

$$\boldsymbol{\dot{P}_i(t) = \Omega(t) P_i(t) + V(t)} \tag{2.14}$$

where $\boldsymbol{\Omega(t) = \dot{R}(t) R(t)^T}$ and $\boldsymbol{V(t) = \dot{D}(t) - \Omega(t) D(t)}$. We see that $\boldsymbol{\Omega(t)}$ is a 3×3 *skew-symmetric* matrix. This can be seen by differentiating the identity $\boldsymbol{R(t) R(t)^T = I}$ with respect to time:

$$\boldsymbol{\dot{R}(t) R(t)^T + R(t) \dot{R}(t)^T = 0} \tag{2.15}$$

and thus, $\boldsymbol{\Omega(t)^T = -\Omega(t)}$. Furthermore, $\boldsymbol{V(t)}$ is a 3D vector.

Because a skew-symmetric matrix product corresponds to a cross product, we can rewrite Eq. (2.14) as:

$$\boldsymbol{\dot{P}_i(t) = \omega(t) \times P_i(t) + V(t)} \tag{2.16}$$

where $\times$ denotes the cross product and $\boldsymbol{\omega(t)}$ is a 3D vector such that $\boldsymbol{\omega(t) \times P_i(t) = \Omega(t) P_i(t)}$. This allows us to interpret $\boldsymbol{\omega(t)}$ and $\boldsymbol{V(t)}$ as the *angular and translational velocity* of the i^{th} 3D point.

This linear algebraic manipulation is important because it allows us to replace the rotation matrix, which has non-linear constraints, with a linear skew-symmetric matrix. Therefore, we can linearly relate the 3D rigid motion of an object to its 2D pixel-wise motion, a relationship that our algorithms exploit.

2.2.2 Estimating Angular and Translational Velocity

In order to estimate the rigid motion in the scene, we first substitute Eq. (2.16) into Eq. (2.10) and Eq. (2.11). For a given time, we can drop the dependence on t and represent the angular and translation velocity simply as $\boldsymbol{\omega}$ and $\boldsymbol{V}$. With $\boldsymbol{\omega} = (\omega_0, \omega_1, \omega_2)^T$ and $\boldsymbol{V} = (V_0, V_1, V_2)^T$, we have:

$$\dot{x_i} = \frac{f}{Z_i} V_0 - \frac{x_i - x_c}{Z_i} V_2 - \frac{(x_i - x_c)(y_i - y_c)}{f} \omega_0 + \left(f + \frac{(x_i - x_c)^2}{f} \right) \omega_1 - (y_i - y_c) \omega_2 \tag{2.17}$$

$$\dot{y}_i = \frac{f}{Z_i} V_1 - \frac{y_i - y_c}{Z_i} V_2 - \left(f + \frac{(y_i - y_c)^2}{f} \right) \omega_0 + \frac{(x_i - x_c)(y_i - y_c)}{f} \omega_1 + (x_i - x_c)\omega_2 \tag{2.18}$$

Both Eq. (2.17) and Eq. (2.18) are linear because in our problem setup, the depth is known in the previous depth map (e.g., Z_i) and its pixel-wise motion (e.g., $\dot{x}_i, \dot{y}_i$) can be determined by finding the corresponding pixels of the consecutive images (e.g., (x_i, y_i) and (x_i', y_i')). Furthermore, for each pixel where Z_i and $(\dot{x}_i, \dot{y}_i)$ are known, we obtain 2 linear equations in 6 unknowns. This means that we can obtain the angular and translational velocity by solving a 6×6 linear system using the data from just *3 pixels*. In practice, more pixels are used to mitigate the noise in the data, but the rigid motion can still be estimated by solving a 6×6 linear system. This can be seen by converting Eq. (2.17) and Eq. (2.18) into matrix form as follows:

$$\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{f}{Z_i} & 0 & -\frac{x_i - x_c}{Z_i} & -\frac{(x_i - x_c)(y_i - y_c)}{f} & f + \frac{(x_i' - x_c)^2}{f} & -(y_i - y_c) \\ 0 & \frac{f}{Z_i} & -\frac{y_i - y_c}{Z_i} & -f - \frac{(y_i - y_c)^2}{f} & \frac{(y_i - y_c)(x_i - x_c)}{f} & (x_i - x_c) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ \omega_0 \\ \omega_1 \\ \omega_2 \end{pmatrix}}_{\boldsymbol{\beta}} = \underbrace{\begin{pmatrix} \vdots \\ \dot{x}_i \\ \dot{y}_i \\ \vdots \end{pmatrix}}_{\boldsymbol{\Delta}} \tag{2.19}$$

where the solution is given by the pseudoinverse:

$$\boldsymbol{\beta} = \left(\boldsymbol{A}^{\boldsymbol{T}} \boldsymbol{A} \right)^{-1} \boldsymbol{A}^{\boldsymbol{T}} \boldsymbol{\Delta} \tag{2.20}$$

With the rigid motion parameters estimated, we can then compute the 3D motion for all of the remaining pixel of a rigid object using Eq. (2.16). As we will show in this thesis, the ability to estimate the 3D motion of a *dense* set of pixels using the data from a *sparse* subset allows our algorithms to obtain depth maps with low latency.

Because our algorithms depend heavily on estimating the rigid motion, an important question to ask is: *how accurate can we estimate the rigid motion with our approach?* We know from [35, 36] that, with the exception of rare *critical surfaces*,

the 2D pixel-wise motion induced by rigid motion is unique. However, our approaches estimate the rigid motion using the 2D optical flow, which is the *apparent* pixel-wise motion, obtained using low-complexity algorithms.

To see how errors in the 2D pixel-wise motion might affect the estimated rigid motion, consider the pixel-wise motion that arises due to a rotation about the y-axis and a translation along the x-axis. This is shown in Figure 2-3, where for a small field-of-view, the pixel-wise motions appear similar. That means that if there are errors in the estimated optical flow, we may not be able to distinguish between these two distinct rigid motions. The same is also true for a rotation about the x-axis and a translation along the y-axis. To understand the limitations of our approach, we analyze the former scenario, which can be easily adapted for the latter case as well.

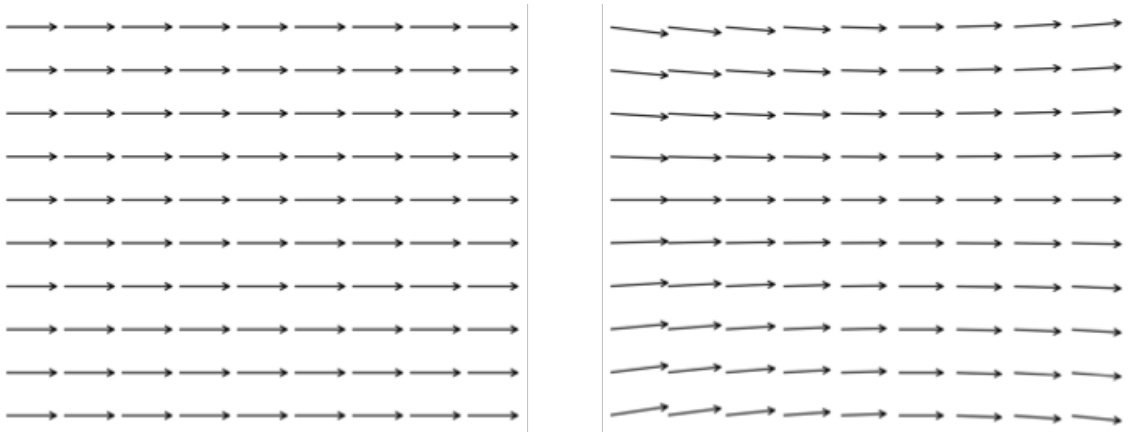


Figure 2-3: **Translation Along x-axis and Rotation About y-axis**: When the field-of-view is small, it hard to distinguish between the pixel-wise motions.

To analyze the case shown in Figure 2-3, we substitute $\boldsymbol{\omega} = (0, \omega_1, 0)^T$ and $\boldsymbol{V} = (V_0, 0, 0)^T$ into Eq. (2.17) and Eq. (2.18). We can then estimate these parameter by solving the following linear system:

$$\begin{pmatrix} \vdots & \vdots \\ \frac{f}{Z_i} & f + \frac{(x_i - x_c)^2}{f} \\ 0 & \frac{(x_i - x_c)(y_i - y_c)}{f} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} V_0 \\ \omega_1 \end{pmatrix} = \begin{pmatrix} \vdots \\ \dot{x}_i \\ \dot{y}_i \\ \vdots \end{pmatrix} \tag{2.21}$$

Our goal is to see how sensitive the parameters, namely V_0 and ω_1, are to errors in optical flow estimates, namely $\dot{x}_i$ and $\dot{y}_i$. We see that if the field-of-view is small (e.g., when f is large), the matrix on the left-hand side of Eq. (2.21) is near singular and thus, has a high condition number. This means that the parameter estimates are sensitive to the noise in the optical flow, which matches our expectation for the case shown in Figure 2-3.

While this scenario seems unfavorable, we know that we can mitigate this in part by using a ToF camera with a larger field-of-view. Even when the field-of-view is fixed, we can also mitigate the impact of erroneous optical flow estimates by estimating the rigid motion with techniques like RANSAC [18].

2.2.3 Estimating the Pose

In some applications, the 3D motion between frames can be sufficiently large such that the approximation in Eq. (2.8) and Eq. (2.9) is invalid. In these scenarios, we must iteratively estimate the pose, which is composed of a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{T}$. To do so we, we first rearrange the terms of Eq. (2.5) and Eq. (2.6) to obtain:

$$(x_i' - x_c)(Z_i + \dot{Z}_i) = f(X_i + \dot{X}_i) \tag{2.22}$$

$$(y_i' - y_c)(Z_i + \dot{Z}_i) = f(Y_i + \dot{Y}_i) \tag{2.23}$$

Given the pose, we can rearrange Eq. (2.7) to obtain:

$$\dot{\boldsymbol{P}}_{\boldsymbol{i}} = (\boldsymbol{R} - \boldsymbol{I})\boldsymbol{P}_{\boldsymbol{i}} + \boldsymbol{T} \tag{2.24}$$

where $\dot{\boldsymbol{P}}_{\boldsymbol{i}} = \boldsymbol{P}_{\boldsymbol{i}}' - \boldsymbol{P}_{\boldsymbol{i}}$ and $\boldsymbol{I}$ is a 3×3 identity matrix. This allows us to use Eq. (2.22) and Eq. (2.23) to obtain the following expressions that relate the pose ($\boldsymbol{R}$ and $\boldsymbol{T}$) of the the i^{th} 3D point ($\boldsymbol{P}_{\boldsymbol{i}}$) to its pixel-wise motion:

$$\Delta x_i(\boldsymbol{R}, \boldsymbol{T}) = \frac{f}{Z_i}\hat{\boldsymbol{x}} \cdot ((\boldsymbol{R} - \boldsymbol{I})\boldsymbol{P}_{\boldsymbol{i}} + \boldsymbol{T}) - \frac{x_i' - x_c}{Z_i}\hat{\boldsymbol{z}} \cdot ((\boldsymbol{R} - \boldsymbol{I})\boldsymbol{P}_{\boldsymbol{i}} + \boldsymbol{T}) \tag{2.25}$$

$$\Delta y_i(\boldsymbol{R}, \boldsymbol{T}) = \frac{f}{Z_i}\hat{\boldsymbol{y}} \cdot ((\boldsymbol{R} - \boldsymbol{I})\boldsymbol{P_i} + \boldsymbol{T}) - \frac{y_i' - y_c}{Z_i}\hat{\boldsymbol{z}} \cdot ((\boldsymbol{R} - \boldsymbol{I})\boldsymbol{P_i} + \boldsymbol{T}) \tag{2.26}$$

where $\Delta x_i(\boldsymbol{R}, \boldsymbol{T})$ and $\Delta y_i(\boldsymbol{R}, \boldsymbol{T})$ represent the components of the 2D pixel-wise motion.

Given the 3D position of N points ($\boldsymbol{P_i}$) and their *estimated* pixel-wise motion ($\dot{x}_i$ and $\dot{y}_i$), we can use Eq. (2.25) and Eq. (2.26) to solve for $\boldsymbol{R}$ and $\boldsymbol{T}$ in a least squares sense by minimizing:

$$\begin{aligned} \min_{\boldsymbol{R},\boldsymbol{T}} \quad & \frac{1}{N}\sum_{i=1}^{N} \left(\dot{x}_i - \Delta x_i(\boldsymbol{R}, \boldsymbol{T})\right)^2 + \left(\dot{y}_i - \Delta y_i(\boldsymbol{R}, \boldsymbol{T})\right)^2 \\ \text{subject to:} \quad & \boldsymbol{R^T R} = \boldsymbol{I} \\ & \det(\boldsymbol{R}) = 1 \end{aligned} \tag{2.27}$$

where the constraints in Eq. (2.27) ensure that $\boldsymbol{R}$ is a proper rotation. We also define the summand of Eq. (2.27) as the residual. We define the residual for the i^{th} pixel as:

$$r_i = \left(\dot{x}_i - \Delta x_i(\boldsymbol{R}, \boldsymbol{T})\right)^2 + \left(\dot{y}_i - \Delta y_i(\boldsymbol{R}, \boldsymbol{T})\right)^2 \tag{2.28}$$

The residual is used in subsequent chapters to estimate the pose robustly.

When estimating the pose, we do not directly estimate the rotation matrix but instead use Rodrigues' Formula [34]. This parameterization is more compact, and we estimate only 4 parameters instead of the 9 parameters of a rotation matrix. With Rodrigues' Formula, we can represent $\boldsymbol{R}$ as follows:

$$\boldsymbol{R} = \boldsymbol{I} + \sin\theta \left[\hat{\boldsymbol{k}}\right]_\times + (1 - \cos\theta)\left[\hat{\boldsymbol{k}}\right]_\times^{\mathbf{2}} \tag{2.29}$$

This describes a rotation of θ radians about an axis, $\hat{\boldsymbol{k}}$. The vector $\hat{\boldsymbol{k}}$ is a unit vector, whose elements form the skew-symmetric matrix, $\left[\hat{\boldsymbol{k}}\right]_\times$, such that $\left[\hat{\boldsymbol{k}}\right]_\times \boldsymbol{P_i} = \hat{\boldsymbol{k}} \times \boldsymbol{P_i}$.

With this representation, we can rewrite Eq. (2.25) and Eq. (2.26) as:

$$\Delta x_i(\hat{\boldsymbol{k}}, \theta, \boldsymbol{T}) = \frac{f}{Z_i}\hat{\boldsymbol{x}} \cdot (\boldsymbol{W}\boldsymbol{P_i} + \boldsymbol{T}) - \frac{x_i' - x_c}{Z_i}\hat{\boldsymbol{z}} \cdot (\boldsymbol{W}\boldsymbol{P_i} + \boldsymbol{T}) \tag{2.30}$$

$$\Delta y_i(\hat{\boldsymbol{k}}, \theta, \boldsymbol{T}) = \frac{f}{Z_i}\hat{\boldsymbol{y}} \cdot (\boldsymbol{W}\boldsymbol{P_i} + \boldsymbol{T}) - \frac{y_i' - y_c}{Z_i}\hat{\boldsymbol{z}} \cdot (\boldsymbol{W}\boldsymbol{P_i} + \boldsymbol{T}) \tag{2.31}$$

where $\boldsymbol{W} = \sin\theta\left[\hat{\boldsymbol{k}}\right]_\times + (1 - \cos\theta)\left[\hat{\boldsymbol{k}}\right]_\times^{\mathbf{2}}$. We can then obtain the pose by solving the following optimization problem:

$$\begin{aligned} \min_{\hat{\boldsymbol{k}},\theta,\boldsymbol{T}} \quad & \frac{1}{N}\sum_{i=1}^{N}\left(\dot{x_i} - \Delta x_i(\hat{\boldsymbol{k}}, \theta, \boldsymbol{T})\right)^2 + \left(\dot{y_i} - \Delta y_i(\hat{\boldsymbol{k}}, \theta, \boldsymbol{T})\right)^2 \\ \text{subject to:} \quad & ||\hat{\boldsymbol{k}}||_2^2 = 1 \end{aligned} \tag{2.32}$$

To obtain $\boldsymbol{R}$ from $\hat{\boldsymbol{k}}$ and θ, we use Eq. (2.29).

Because the objective function in Eq. (2.32) is non-linear, we must solve it iteratively and update the variables incrementally. To determine these updates, we first linearize $\boldsymbol{W}$, the only non-linear term in Eq. (2.30) and Eq. (2.31), about $\theta = 0$. We have:

$$\boldsymbol{W} = \sin(\delta\theta)\left[\hat{\boldsymbol{k}}\right]_\times + (1 - \cos(\delta\theta))\left[\hat{\boldsymbol{k}}\right]_\times^{\mathbf{2}} \approx \delta\theta\left[\hat{\boldsymbol{k}}\right]_\times \tag{2.33}$$

where the right most term is obtained by using the small angle approximation. This approximation is important because it allows us to remove the constraint on $\hat{\boldsymbol{k}}$ in the objective function in Eq. (2.32) and treat $\boldsymbol{W}$ as an arbitrary skew-symmetric matrix (since $\delta\theta$ and $\hat{\boldsymbol{k}}$ are independent of each other). Therefore, we can solve for the iterative update using *linear* least squares.

Denoting the entries of $\boldsymbol{W}$ and $\boldsymbol{T}$ as:

$$\boldsymbol{W} = \begin{pmatrix} 0 & -W_2 & W_1 \\ W_2 & 0 & -W_0 \\ -W_1 & W_0 & 0 \end{pmatrix} \qquad \boldsymbol{T} = \begin{pmatrix} T_0 \\ T_1 \\ T_2 \end{pmatrix} \tag{2.34}$$

we can rewrite the expressions in Eq. (2.30) and Eq. (2.31) to obtain:

$$\dot{x_i} = \frac{f}{Z_i}T_0 - \frac{x_i' - x_c}{Z_i}T_2 - \frac{(x_i' - x_c)Y_i}{Z_i}W_0 + \left(f + \frac{(x_i' - x_c)X_i}{Z_i}\right)W_1 - \frac{fY_i}{Z_i}W_2 \tag{2.35}$$

$$\dot{y_i} = \frac{f}{Z_i}T_1 - \frac{y_i' - y_c}{Z_i}T_2 - \left(f + \frac{(y_i' - y_c)Y_i}{Z_i}\right)W_0 + \frac{(y_i' - y_c)X_i}{Z_i}W_1 + \frac{fX_i}{Z_i}W_2 \tag{2.36}$$

We see that Eq. (2.35) and Eq. (2.36) are linear in the parameters that describe $\boldsymbol{W}$ and $\boldsymbol{T}$. Therefore, for each pixel whose optical flow and depth are known, we obtain 2 linear equations in 6 unknowns. Similar to the case where we estimated the angular and translational velocity, we can solve for the incremental update and pose by solving a 6×6 linear system using the data from just *3 pixels*. In practice, more pixels are used to mitigate the noise in the optical flow and depth values, but the incremental update is still equivalent to solving a 6×6 linear system. This can be seen by converting Eq. (2.35) and Eq. (2.36) into matrix form as follows:

$$\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{f}{Z_i} & 0 & -\frac{x_i'-x_c}{Z_i} & -\frac{(x_i'-x_c)Y_i}{Z_i} & f+\frac{(x_i'-x_c)X_i}{Z_i} & -\frac{fY_i}{Z_i} \\ 0 & \frac{f}{Z_i} & -\frac{y_i'-y_c}{Z_i} & -f-\frac{(y_i'-y_c)Y_i}{Z_i} & \frac{(y_i'-y_c)X_i}{Z_i} & \frac{fX_i}{Z_i} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} T_0 \\ T_1 \\ T_2 \\ W_0 \\ W_1 \\ W_2 \end{pmatrix}}_{\boldsymbol{\beta}} = \underbrace{\begin{pmatrix} \vdots \\ \dot{x}_i \\ \dot{y}_i \\ \vdots \end{pmatrix}}_{\boldsymbol{\Delta}} \tag{2.37}$$

where the solution is again given by the pseudoinverse as in Eq. (2.20). With the derivation of the incremental update, we summarize how we estimate $\boldsymbol{R}$ and $\boldsymbol{T}$ in Algorithm 1.

2.3 Summary of Notation

Before we conclude this chapter, we summarize the notation we will use for the remainder of this thesis. We denote (RGB or IR) images and depth maps as $I(x, y)$ and $D(x, y)$, respectively. When appropriate, we use subscripts to indicate the temporal relationship between the images and depth maps. For example, we refer to the previous image and depth map pair as $I_{t-1}(x, y)$ and $D_{t-1}(x, y)$, respectively, and the current pair as $I_t(x, y)$ and $D_t(x, y)$, respectively. We also define frames to be the images and depth maps acquired at the same time instance.

We assume that both the digital and ToF cameras are calibrated and have the

Algorithm 1 Estimating $\boldsymbol{R}$ and $\boldsymbol{T}$

input: Corresponding pixels (x_i, y_i) and (x'_i, y'_i); depth Z_i; principal distance f and point (x_c, y_c); number of iterations N_I
output: $\boldsymbol{R}$ and $\boldsymbol{T}$

$\boldsymbol{R} \leftarrow \boldsymbol{I}$
$\boldsymbol{T} \leftarrow \boldsymbol{0}$
Compute $\boldsymbol{P_i}$ using Eq. (2.2) for $i = 1, 2, \ldots N$
counter $\leftarrow 1$
repeat
    $\boldsymbol{P_i} \leftarrow \boldsymbol{R}\boldsymbol{P_i} + \boldsymbol{T}$ for $i = 1, 2, \ldots N$
    Update (x_i, y_i) using $\boldsymbol{P_i}$ and Eq. (2.3) for $i = 1, 2, \ldots N$
    $\dot{x}_i \leftarrow x'_i - x_i$ and $\dot{y}_i \leftarrow y'_i - y_i$ for $i = 1, 2, \ldots N$
    Form $\boldsymbol{A}$ and $\boldsymbol{\Delta}$ in Eq. (2.37) using $\boldsymbol{P_i}$, $\dot{x}_i$, and $\dot{y}_i$
    $(\boldsymbol{\delta T^T}, \boldsymbol{\delta W^T})^{\boldsymbol{T}} \leftarrow (\boldsymbol{A^T A})^{-1} \boldsymbol{A^T \Delta}$
    $\boldsymbol{\delta R} \leftarrow \boldsymbol{I} + \frac{\sin(||\boldsymbol{\delta W}||_2)}{||\boldsymbol{\delta W}||_2}[\boldsymbol{\delta W}]_\times + \frac{1-\cos(||\boldsymbol{\delta W}||_2)}{||\boldsymbol{\delta W}||_2^2}[\boldsymbol{\delta W}]_\times^2$
    $\boldsymbol{R} \leftarrow \boldsymbol{\delta R}\boldsymbol{R}$ and $\boldsymbol{T} \leftarrow \boldsymbol{\delta R}\boldsymbol{T} + \boldsymbol{\delta T}$
    counter $\leftarrow$ counter $+ 1$
until counter $= N_I$

same intrinsic parameters. We denote f is the principal distance (focal length), and (x_c, y_c) as the principal point (center). We refer to the 3D coordinate of the i^{th} point as $\boldsymbol{P_i} = (X_i, Y_i, Z_i)^T$, and the 2D coordinate of its projection as $\boldsymbol{p_i} = (x_i, y_i)^T$. These quantities are defined in Eq. (2.2) and Eq. (2.3), respectively. As done in this chapter, we will continue to distinguish vectors and matrices from scalars by bolding the former.

To simplify notation, we also define the projection operator, $\pi(\boldsymbol{P_i}) = \boldsymbol{p_i}$, to map a 3D point to its pixel coordinate. We also make use of standard linear algebra notation. We denote $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}$ as the unit vectors oriented along the 3D coordinate axes. We also use standard operations like the dot product, e.g. $\boldsymbol{P_i} \cdot \boldsymbol{P_j}$, and the cross product, $\boldsymbol{P_i} \times \boldsymbol{P_j}$.

Chapter 3

Depth Map Estimation for Rigid Objects

3.1 Introduction

In this chapter, we describe how we use concurrently collected RGB images to reduce the usage of a time-of-flight (ToF) camera and estimate new depth maps for rigid objects. This is shown in Figure 3-1, where our algorithm sequentially estimates new depth maps without turning on the ToF camera. This is especially useful for tasks that assume that their surrounding environments are static, where the changes from frame to frame are due to ego-motion (e.g., the motion of the user or robot). In these scenarios, we can treat the entire scene as a single rigid object. These tasks include simultaneous localization and mapping (SLAM) [30], object detection and avoidance [31], and object manipulation [17]. These tasks all benefit from using depth information and are central to applications that range from augmented reality to robotic navigation. While the assumption of rigidity may seem limiting, our approach only requires the local environment that the ToF camera *can sense* to be rigid. For many mobile applications, this is a reasonable assumption because the range of the ToF camera is limited.

The assumption of rigidity is also important because it allows us to efficiently estimate new depth maps using the optical flow at a sparse set of pixels and a single,

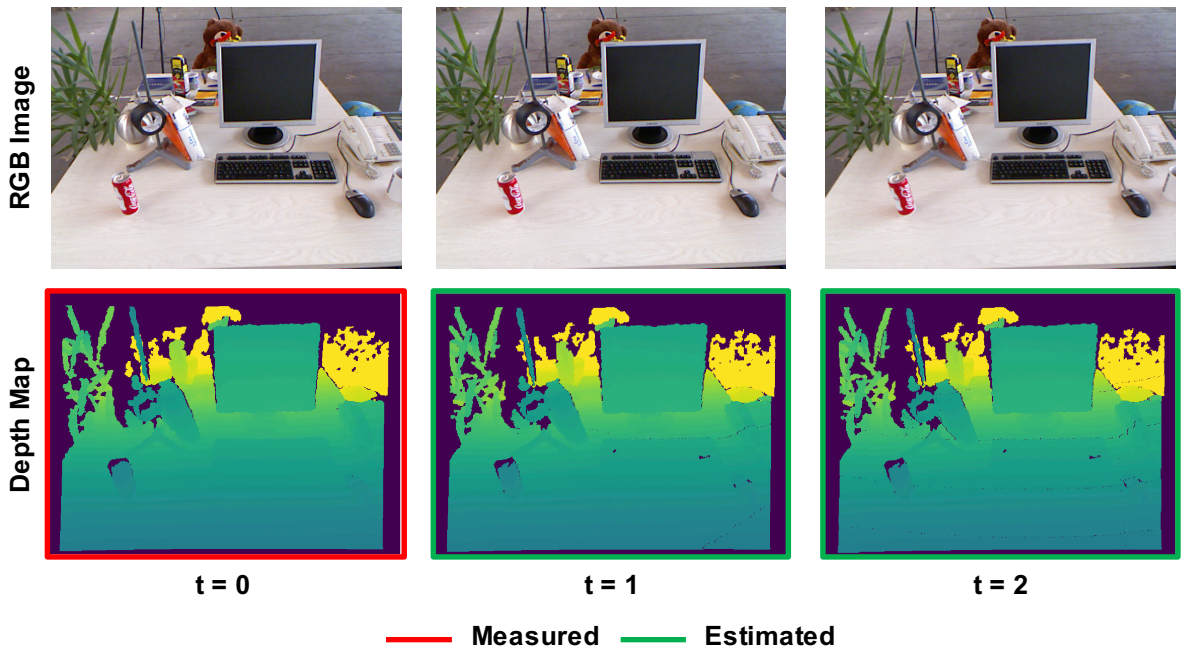


Figure 3-1: **Depth Map Estimation**: We estimate causal depth maps using concurrently collected images and a previously measured depth map. The ToF camera is only used when an accurate depth map cannot be estimated.

previously measured depth map. This not only allows us to reduce the sensor power, but also reduce the *overall system power* for depth sensing by enabling our algorithm to estimate accurate depth maps in real-time on a low power embedded platform. Our contribution, therefore, is an optimized algorithm that combines computationally efficient techniques to obtain an accurate and dense depth map with minimal latency. Our approach balances the usage of the ToF camera, the computational costs of the algorithm, and the quality of the estimated depth map. In particular, we present the following:

- We introduce an algorithm that lowers the usage of the ToF camera while maintaining the acquisition of depth maps. To obtain new depth maps, our algorithm estimates the 3D motion in the scene and uses it to update a previously measured depth map.

- We reduce the computation required to estimate the 3D motion of every pixel by estimating the rigid motion, or pose, instead. We show that it is possible

to obtain an accurate depth map by using the pose, which can be estimated with the optical flow determined by a block matching heuristic on a sparse and uniformly-spaced grid. This is essential for our approach to run in real-time on a low power embedded platform, which also allows the overall system power for depth sensing to be reduced.

- We propose a policy that adaptively enables the ToF camera when an accurate depth map cannot be estimated. This is crucial because it is not always possible to estimate accurate optical flow (especially with limited computational resources), which is required for our technique.

To demonstrate the efficiency of our approach and quantify the reduction in power, we implement our algorithm on the ODROID XU-3 board [26] using only the low power CPUs. We demonstrate a real-time implementation that reduces the overall system power for depth sensing and show that the depth maps can be used in a real augmented reality pipeline. In addition to estimating depth maps temporally, we also show how our algorithm can be used to infill depth spatially and extend the range of a ToF camera.

This chapter is based on our work in [62,64] and is organized as follows. In Section 3.2, we describe other related approaches that use images to aid in the estimation of depth maps. This is followed by a description of our approach in Section 3.3, where we describe how we robustly estimate the rigid motion across consecutive frames and use the rigid motion to obtain a new depth map. In Section 3.4, we evaluate our algorithm on a variety of RGB-D datasets, where we also analyze the impact of our design choices and compare our algorithm to other techniques. While quantitative evaluations are important, we also show that our depth maps can be used in real applications, and in Section 3.5, we show how our estimated depth maps can be used for augmented reality. To estimate the reduction in the power for depth sensing, we quantify the overall system power of our approach in Section 3.6. In Section 3.7, we show how our algorithm can also be used to infill depth spatially. Finally, we summarize the key impact of our work in Section 3.8.

3.2 Background

The idea of using RGB images to help estimate new depth maps has been explored in many applications. Given its breadth, we only focus on techniques that have similar problem setups, namely those that estimate new depth maps temporally using concurrently collected RGB images and previously measured depth maps (Section 3.2.1) and those that only use consecutive RGB images (Section 3.2.2).

3.2.1 Temporal Depth Map Estimation

Here, we describe techniques that use RGB images to temporally estimate new depth maps from previously measured ones for applications where both depth maps and RGB images are concurrently collected. Choi *et al.* [11] addresses the fact that depth maps are typically acquired at lower frame rates than RGB images for many applications. To equalize the frame rates, the authors applied bidirectional block matching algorithms to estimate the optical flow between RGB images without any corresponding depth maps and those with it. These optical flow vectors are used to identify the depth blocks that are averaged to form a new depth map. Similarly, Wang *et al.* [89] and Zhang *et al.* [95] also estimate depth maps between frames that have both RGB images and depth maps available using block matching algorithms. Wang *et al.* [89] selects the depth block from either the preceding or future depth map based on the edges of the corresponding RGB image blocks. Zhang *et al.* [95] estimates depth by performing a weighted average guided by the underlying texture in the RGB images. All of these approaches use block matching algorithms to obtain dense optical flow fields, but this process is computationally expensive. To reduce the complexity, Li *et al.* [50] reuse the motion vectors generated in compressed video to accelerate the process of depth map estimation.

Unfortunately, we cannot directly use these techniques to obtain new depth maps because computing a dense optical flow field has *prohibitively high latency* on embedded processors. For example, OpenCV's implementation of dense optical flow [16] runs at 0.88 frames per second on our embedded device (ODROID-XU3 [26]) for

640×480 images. For applications like 3D video frame upsampling, which can be performed offline, this is not necessarily a problem, but for applications like robotic navigation, these approaches are unsuitable because the underlying applications are sensitive to latency. Furthermore, most of these approaches also estimate depth maps by using the depth from preceding and *future frames*. This is not possible for real-time applications, where we require the estimation of depth maps to be causal. As we show in Section 3.3, our algorithm uses the assumption of rigidity to significantly reduce the computation required to obtain low latency and causal depth maps.

3.2.2 Pose Estimation and Structure-from-Motion

As stated in Section 3.1, our algorithm estimates the rigid motion from frame to frame. This motion can be represented by the relative pose, which is composed of a rotation and translation. A common way to estimate the pose exploits epipolar geometry and uses the pixel-wise correspondences between consecutive RGB images (which can be trivially obtained using the optical flow) to obtain an intermediate quantity known as the essential matrix, which can then factored to obtain the rotation and translation [28]. Depending on the number of correspondences, the essential matrix can be estimated using techniques that range from performing a singular value decomposition (8 correspondences) [54] to finding the roots of a tenth order polynomial (5 correspondences) [60].

One potential benefit of this approach is that it only requires RGB images to obtain the pose, although the estimated translation is known only to scale (the magnitude of the translation vector is not known). Furthermore, once the pose is obtained, relative depth can also be estimated by triangulating the corresponding pixels. These techniques are known as structure-from-motion (SfM) [14, 81, 84, 91], and we refer the interested reader to a comparison [5] of popular and state-of-the-art pipelines. Unfortunately, one drawback of these approaches is that they typically only estimate depth at a sparse set of keypoints. This is problematic for applications like obstacle avoidance, which require dense depth maps. Furthermore, these techniques also only estimate relative depth. Unlike the SfM techniques, our approach estimates the pose

using the method described in Section 2.2.3, which obtains the rotation and absolute translation with fewer correspondences than the essential matrix-based approaches. We then use this pose to update a previously measured depth map to obtain a *dense* depth map.

3.3 Proposed Algorithm

Our proposed algorithm takes as input consecutive RGB images and a previous depth map and outputs a new one as shown in Figure 3-2. Our proposed technique is computationally efficient, and we highlight our design choices so that our algorithm can run in real-time on an embedded platform. We also describe our strategy to adaptively use the ToF camera when an accurate depth map cannot be estimated. The pipeline of our approach is shown in Figure 3-3.

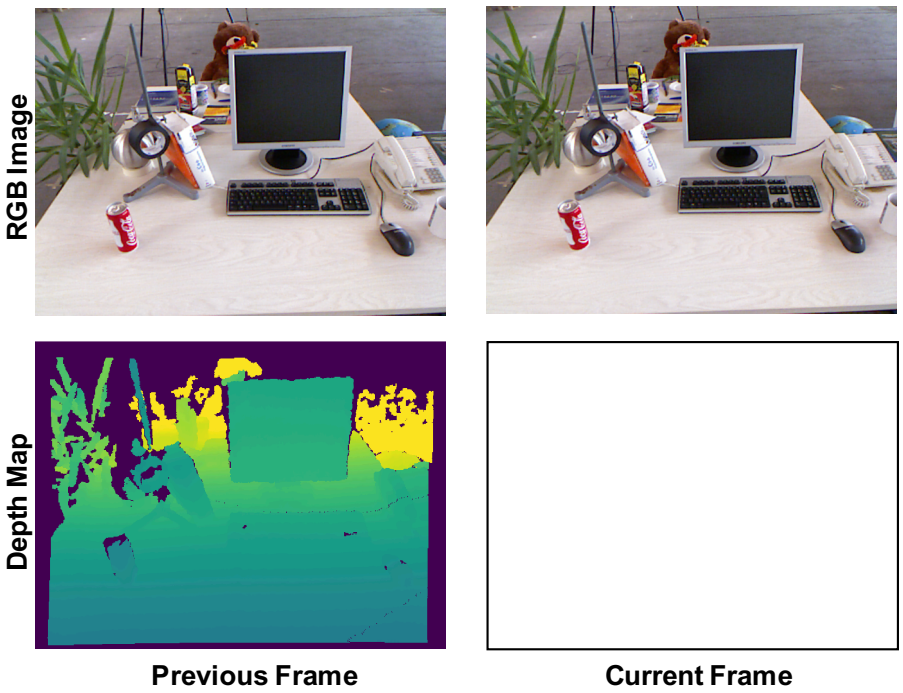


Figure 3-2: **Problem Setup**: Our algorithm estimates the pose and a new depth map using two consecutive RGB images and a previous depth map.

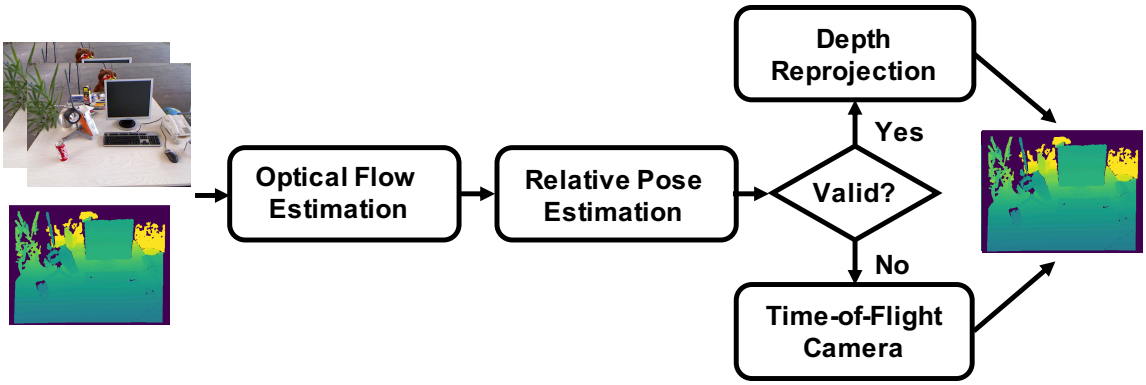


Figure 3-3: **Depth Map Estimation Pipeline**: Our algorithm estimates the pose by using the optical flow across the consecutive images and the previous depth measurements. It then uses this pose to reproject the previous depth map to obtain a new one. When a reliable pose cannot be estimated, we use the ToF camera to obtain a new depth map instead.

3.3.1 Optical Flow Estimation

As shown in Figure 3-3, we begin by first estimating the optical flow between the consecutive RGB images using the three step search (TSS) algorithm [45]. The TSS algorithm obtains the optical flow for a block of pixels in the RGB image by searching for the block in the next RGB image that minimizes a cost function. However, instead of an exhaustive search, the TSS algorithm only considers select locations to reduce computation.

Our decision to use the TSS algorithm is motivated by its runtime on an embedded platform. We compare the runtime of the TSS algorithm to the commonly used Lucas Kanade algorithm [56] by profiling both approaches on the ODROID-XU3 board [26], which is an embedded platform that is representative of the compute resources available on mobile devices. We use the low power Cortex-A7 cores to compute the optical flow for 640×480 images. In our experiments across a variety of different datasets (Section 3.4), we find that using the TSS algorithm with 15×15 blocks with a step size of 8 on the pixels of a uniformly-spaced, 12×12 grid enabled our algorithm to estimate low latency and accurate depth maps. For the same set of pixels, we use the Lucas Kanade algorithm with 15×15 blocks and 3 pyramid levels so that we have a similar search area. On average, we find that the TSS algorithm

requires 13 ms whereas the Lucas Kanade algorithm requires 51 ms. We also profile the time required to identify corners. We found that the Harris corner detector [27] requires 120 ms, which is intolerable for real time applications, whereas the time to locate the pixels on a uniform grid is negligible. We summarize these runtimes in Table 3.1.

Algorithm	**Runtime (ms)**	**Frame Rate (FPS)**
Three Step Search	13	76.9
Lucas Kanade	51	19.6
Harris Corner	120	8.3

Table 3.1: **Runtime Comparisons**: We profile our design choices on the ODROID-XU3 board [26]. We opt to use the TSS algorithm to ensure our implementation can estimate depth maps in real-time.

However, as shown in Figure 3-4, one drawback of using the TSS algorithm is that our optical flow estimates can be inaccurate. In the next section, we show how we can mitigate this to robustly estimate the pose and depth map. With the pose estimated, in addition to obtaining a new depth map, we can also correct the optical flow field as shown in Figure 3-5.

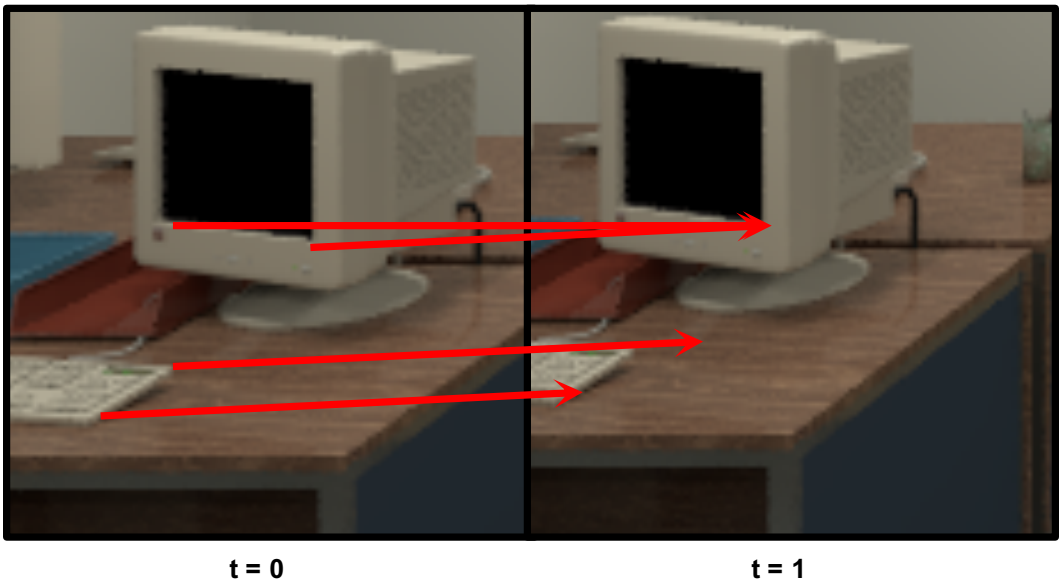


Figure 3-4: **Optical Flow from TSS**: We show examples of optical flow vectors estimated using the TSS algorithm.

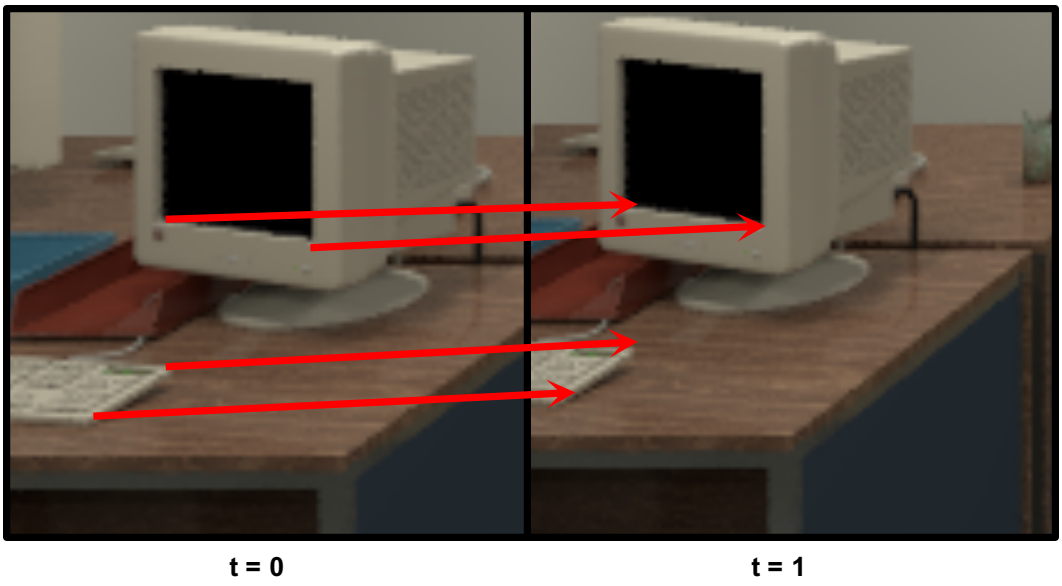


Figure 3-5: **Optical Flow from Pose**: We show examples of the optical flow vectors computed using the estimated pose.

3.3.2 Relative Pose Estimation

With the optical flow estimated, we can then estimate the relative pose using Algorithm 1 as described in Section 2.2.3. However, using the optical flow directly is problematic because it can be different from the underlying motion field [34]. Image sensor noise, occlusions, and the algorithm used to estimate the optical flow affect the accuracy of the estimated pose. Furthermore, because the optical flow is estimated using image intensities, it can be different from the underlying motion field even in the absence of these issues. In regions with uniform intensity, for example, the optical flow would be zero even when the underlying motion field is not. Moreover, the previous depth map can also be affected by sensor noise in addition to irregularities that arise from multipath reflections, specular reflections, and interference [25]. Because Algorithm 1 directly uses these depth values, these errors can also adversely affect the estimated pose.

While these errors are in part mitigated by our least squares formulation in Eq. (2.32), we need a mechanism to distinguish accurate optical flow and depth from erroneous ones because the squared penalty in our formulation is not robust against

large errors in the optical flow. This is possible when the pose is known, and we can distinguish the accurate optical flow estimates, or inliers, from the erroneous ones, or outliers, because the former satisfy Eq. (2.25) and Eq. (2.26). This insight suggests that we estimate the pose using RANSAC [18].

We proceed by randomly selecting the optical flow estimates and its corresponding depth to obtain an initial pose hypothesis. We use 3 optical flow estimates, which is the minimum required to estimate pose using our technique, to minimize the likelihood of choosing an outlier. To judge the quality of the pose hypotheses, we relax the requirement that the pose must satisfy Eq. (2.25) and Eq. (2.26) for all of the inliers and instead compute the residual error for each optical flow estimate using Eq. (2.28). If the number of optical flow estimates with low residual errors, which is determined by the RANSAC threshold, exceed a fraction of the total number of estimates, we then re-estimate the pose using only these inliers. We repeat this procedure and select the candidate pose with the lowest mean residual error. When there are no candidates, we enable the ToF camera to acquire a new depth map. This adaptive control of the ToF camera, which is a beneficial side effect of using RANSAC, allows for accurate depth maps to be obtained. We summarize our approach in Algorithm 2.

To show that Algorithm 2 can mitigate the impact of errors in the depth and the optical flow, we first simulate the idealized depth and optical flow for a given pose and corrupt a subset of them. To reflect the fact that our approach uses the TSS algorithm, we also round each optical flow vector to the nearest integer displacement. We then estimate the pose with and without RANSAC and compare it to the pose we used to simulate the data with using the root mean squared error (RMSE) of the translation as defined in [85]. In Table 3.2, we see that RANSAC substantially lowers the RMSE of the translation across all scenarios. This is significant because RANSAC allows us to use the optical flow obtained from the TSS algorithm to estimate the pose accurately.

In our experiments across a variety of datasets (Section 3.4), we use 30 RANSAC iterations and set the RANSAC threshold to 4 and accept a pose hypothesis if the size

Algorithm 2 Adaptive Pose Estimation

input: Optical flow $(\Delta x_i, \Delta y_i)$, depth (Z_i), and RANSAC parameters (No. of iterations, thresh, and min. size)
output: Pose ($\boldsymbol{R}$ and $\boldsymbol{T}$) or signal to use ToF camera

repeat ▷ Get the inlier set
    Randomly sample 3 optical flow vectors and its depth and estimate the pose using Algorithm 1
    Compute the residuals, r_i, as defined in Eq. (2.28)
    Get inlier set, $\mathcal{I} = \{i : r_i < \text{thresh}\}$
    Retain $\mathcal{I}$ with lowest mean residual; $|\mathcal{I}| >$ min. size
until End of RANSAC

if $|\mathcal{I}| = 0$ **then** ▷ Get pose or depth map
    Use the ToF camera
else
    Estimate the pose using Algorithm 1 with $\mathcal{I}$
end if

Depth	Optical Flow	Reduction (%)
✓		68.0
	✓	59.4
✓	✓	45.1

Table 3.2: **Impact of RANSAC**: We present the reduction in the RMSE obtained using RANSAC when there is noise in the depth measurements, the optical flow, and in both.

of its inlier set is at least 10% of the number of optical flow estimates. When obtaining the initial pose, we perform 1 iteration of Algorithm 1, which is similar to estimating the angular and translation velocity (Section 2.2.2). We then estimate the pose using the inlier set by performing 3 iterations of Algorithm 1. As we describe in Section 3.4.1, this accounts for a small fraction of the overall run-time of our algorithm. This is essential to obtaining accurate depth maps in real-time.

3.3.3 Depth Reprojection

Once the pose is estimated, we obtain a new depth map by applying the pose to each 3D point in the first depth map and projecting its depth, or its z-coordinate, to an

image. For every pixel in the first depth map, we first compute its 3D point, $\boldsymbol{P_i}$, using Eq. (2.2). The reprojected depth map is then obtained as follows:

$$D\left[f\frac{\hat{\boldsymbol{x}}\cdot(\boldsymbol{RP_i}+\boldsymbol{T})}{\hat{\boldsymbol{z}}\cdot(\boldsymbol{RP_i}+\boldsymbol{T})}+x_c, f\frac{\hat{\boldsymbol{y}}\cdot(\boldsymbol{RP_i}+\boldsymbol{T})}{\hat{\boldsymbol{z}}\cdot(\boldsymbol{RP_i}+\boldsymbol{T})}+y_c\right]=\hat{\boldsymbol{z}}\cdot(\boldsymbol{RP_i}+\boldsymbol{T}) \tag{3.1}$$

where D represents the depth map whose entries are indexed by its x- and y-coordinates and $\boldsymbol{R}, \boldsymbol{T}$ is the pose. If multiple points are mapped onto the same pixel location, we retain the smallest depth value.

When more than one depth map is predicted consecutively, we obtain a new depth map by reprojecting the last measured depth map. To do so, we update the pose accordingly. Let $\boldsymbol{R_c}$ and $\boldsymbol{T_c}$ represent the current pose that is estimated using the previously estimated depth map. We also assume that the previously estimated depth map was obtained by reprojecting the last measured depth map using $\boldsymbol{R_{t-1}}$ and $\boldsymbol{T_{t-1}}$. Then, the pose which we now use to reproject the previously measured depth map, denoted as $\boldsymbol{R_t}$ and $\boldsymbol{T_t}$, is:

$$\boldsymbol{R_t}=\boldsymbol{R_c}\boldsymbol{R_{t-1}} \qquad \boldsymbol{T_t}=\boldsymbol{T_c}+\boldsymbol{R_c}\boldsymbol{T_{t-1}} \tag{3.2}$$

The resulting depth map contains depth estimates for pixels that correspond to the overlapping field-of-views between the image where the last depth map was measured and the current image. It should be noted that without any additional post-processing, this method also introduces artifacts as shown in Figure 3-6. These holes arise because the pixels belonging to the same object are treated independently and are not constrained to be contiguous after reprojection and because regions that were previously occluded have been uncovered. While reverse warping would eliminate these holes, it also erroneously infills the previously occluded regions. We want to avoid this, especially as we predict many depth maps consecutively, in the event where the previously occluded region has a different depth from its surroundings. We confirm this by applying our algorithm to sequences from the TU Munich RGB-D dataset [85] and find that reverse warping increases the overall mean relative error (as defined in Section 3.4.3) by 17.4% compared to our approach. Furthermore, if

the application needs the depth in the previously occluded regions, this could serve as another signal to use the ToF camera.

One potential way to remove the first type of holes is by applying a median filter with a small kernel size to the resulting depth map as shown in Figure 3-6. While this may give inconsistent behaviors at depth boundaries, we find in our experiments with the TU Munich RGB-D dataset, the overall mean relative error remains unchanged. However, as our computational resources are limited, we ignore this additional step because these types of holes are minimal, accounting for less than 3% of the estimated pixels while imposing an additional 20 ms overhead. In the next section, we see that this is intolerable for real-time performance.

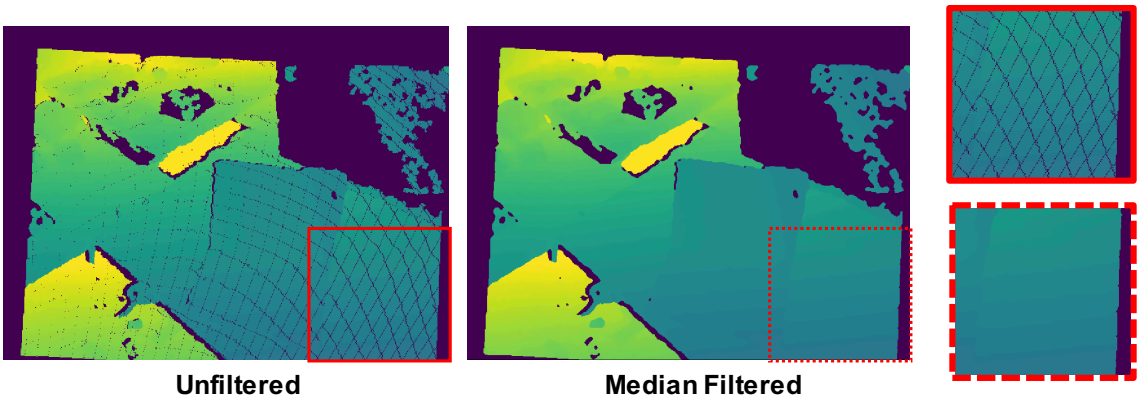


Figure 3-6: **Reprojected Depth Maps**: The reprojected depth maps have artifacts where depth is not available. While a median filter can infill these regions, we ignore this post-processing step because the holes constitute a small portion of the depth map.

3.4 Algorithm Evaluation

3.4.1 Implementation

We implement our algorithm on the ODROID XU-3 board [26], which is an embedded platform with an Exynos 5422 processor. The Exynos processor is used in the Samsung Galaxy S5 [78] and is representative of the compute power available on mobile devices. Our implementation uses the low power Cortex-A7 cores of the board and

outputs 640×480 depth maps in real time, or 30 frames per second (FPS). To achieve this frame rate, we parallelize our computation across the 4 Cortex-A7 cores. We use the parameter settings described in Section 3.3 and the OpenCV library whenever possible.

As shown in Figure 3-7, most of the time of our implementation is spent on estimating the optical flow and reprojecting the depth map. This figure further justifies our decision to use the TSS algorithm. Since the time required to reproject a depth map is fixed, we are limited in what we can allocate to obtain the optical flow if we want to estimate depth maps at 30 FPS. We discuss the impact of this decision on the accuracy of the estimated depth maps in Section 3.4.5. This figure also shows that when only the pose is required, which is the case for SLAM, our algorithm can run at nearly 58 FPS.

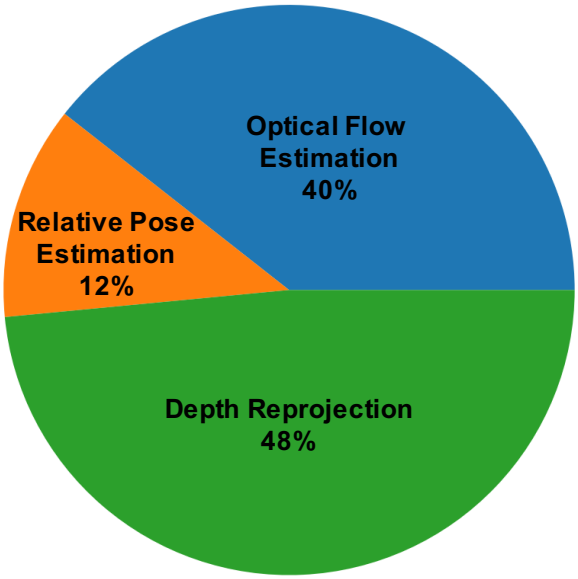


Figure 3-7: **Runtime Breakdown**: We profile the implementation of our algorithm on the ODROID-XU3 [26] board, which produces 640×480 depth maps at 30 FPS. Because the time to reproject a new depth map is fixed, we work to reduce the computation time required to estimate the optical flow.

3.4.2 Dataset

We evaluate our algorithm on RGB-D datasets used to benchmark SLAM, visual odometry, 3D reconstruction, and navigation algorithms. These tasks are relevant

for many mobile applications, and the images and depth maps are representative of what our approach will encounter. We adapt these datasets to test our approach by using consecutive images and select depth maps to predict new ones, which we then compare to that in the dataset. For our experiments, we use the provided intrinsic parameters and tools to synchronize the images with the depth maps for each dataset.

We use sequences from the following datasets: TU Munich RGB-D [85], NYU Depth V2 [83], Indoor RGB-D [80], CoRBS [90], and ICL-NUIM [24]. These datasets contain 640×480 RGB images and depth maps and most are collected at 30 FPS.

3.4.3 Methodology

We apply our algorithm to the first 100 frames of the sequences in each dataset. We quantify the accuracy of the depth maps using:

- **Mean Relative Error (MRE)**: This metric is defined in Eq. (1.10) and penalizes a unit error more so up close than further away, which is an appropriate metric for applications (e.g., robotic navigation) that need to react quickly to changes in its immediate surroundings. It also allows us to compare the performance of our algorithm across datasets with different dynamic ranges.

To highlight the different dynamic ranges, we also compute the following error metrics:

- **Mean Absolute Error (MAE)**: This is defined as $\frac{1}{N}\sum_{j=1}^{N}|Z_j - \hat{Z}_j|$ and presented in centimeters.
- **Root Mean Squared Error (RMSE)**: This is defined as $\sqrt{\frac{1}{N}\sum_{j=1}^{N}(Z_j - \hat{Z}_j)^2}$ and presented in centimeters.

Because our algorithm uses the ToF camera adaptively, we also quantify the frequency at which it is used using:

- **Duty Cycle (DC)**: This is equal to $\sum_{i=1}^{100}\mathbb{1}(i)$, where $\mathbb{1}(i)$ equals 1 if the i^{th} depth map is obtained using the ToF camera and 0 if the depth map is estimated instead. The DC is presented as a percentage.

To reduce the power required to obtain accurate depth maps, our goal is to lower the ToF camera's duty cycle while maintaining a baseline accuracy. In our analysis, we focus on the MRE because it allows us to compare the performance of our algorithm across datasets that have different ranges of depth. Therefore, for each dataset, we set the threshold parameter in Algorithm 2 to achieve a median MRE of approximately 1% across its sequences in order to measure its duty cycle.

3.4.4 Results

We summarize the performance of our algorithm for each dataset in Table 3.3, where we compute the median of each error metric across the depth maps. Examples of the estimated depth maps are shown in Figure 3-8. Across the datasets, we achieve a median MRE of 0.96% and a median duty cycle of 15.0%. In Table 3.3, we see that the duty cycle for Indoor RGB-D [80] is higher than that of the other datasets. This is expected because this dataset contains sequences of a robot moving abruptly in a sparsely textured environment. Furthermore, this shows that our technique can adapt to and still reduce the usage of the ToF camera in these challenging scenarios.

Dataset	**MRE (%)**	**MAE (cm)**	**RMSE (cm)**	**DC (%)**
TU Munich RGB-D	0.96	2.27	7.63	16.0
NYU Depth V2	0.95	4.04	9.01	10.0
Indoor RGB-D	1.03	2.11	7.54	33.0
CoRBS	1.04	1.79	8.98	15.0
ICL-NUIM	0.67	2.04	5.65	10.0
Mean	0.93	2.45	7.76	16.8
Median	0.96	2.11	7.63	15.0

Table 3.3: **Algorithm Evaluation**: We summarize the MRE, MAE, RMSE and DC that our algorithm achieves.

Because different applications have different accuracy requirements for depth maps, we also quantify the tradeoff between the duty cycle and the MRE for our approach. To do so, we vary the RANSAC threshold in Algorithm 2 that determines if an optical flow estimate is an inlier. In our pipeline, we expect that a lower threshold, which assumes accurate optical flow estimates, will result in depth maps with a lower MRE

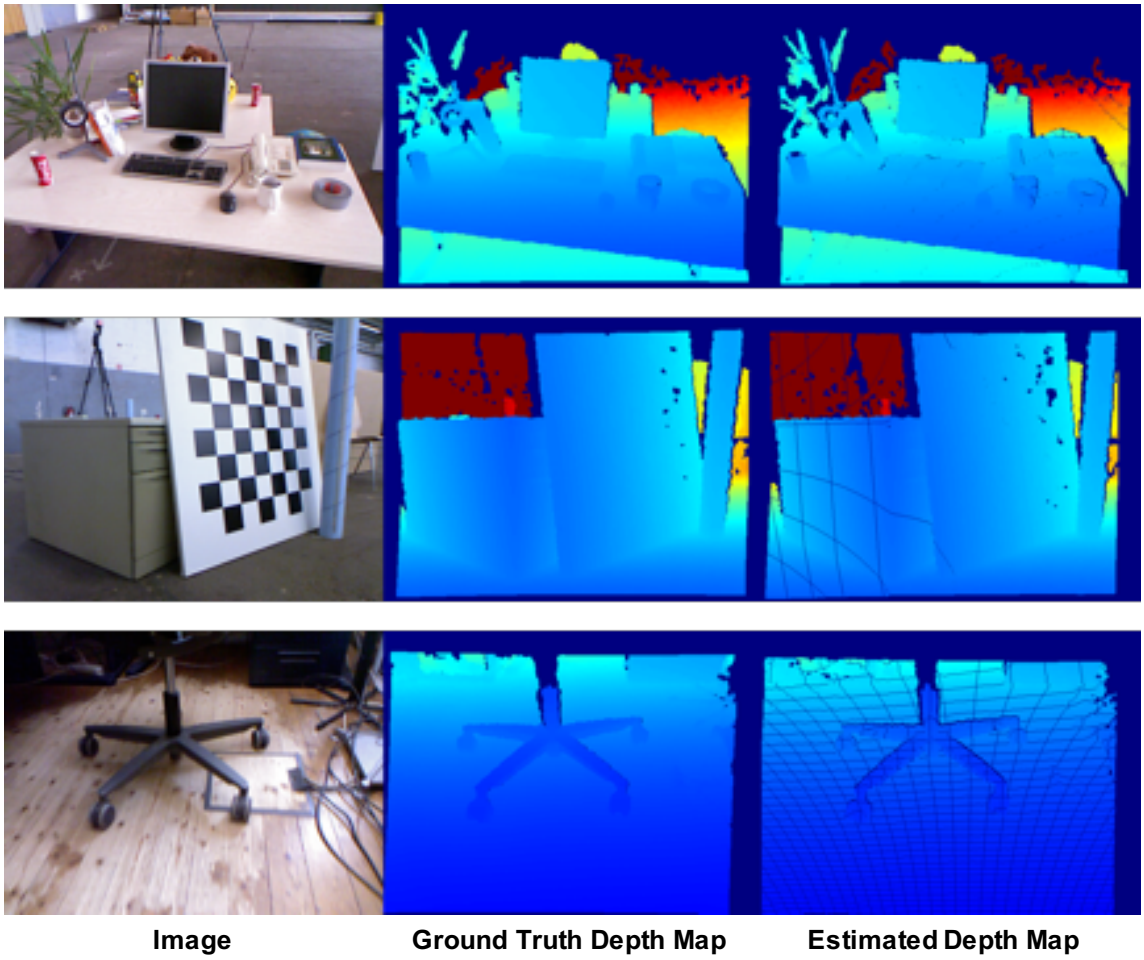


Figure 3-8: **Estimated Depth Maps**: We show the estimated depth maps for select sequences in [85]. A video can be found at `https://youtu.be/47L4xebYHTI`.

but also a higher duty cycle because the TSS algorithm cannot consistently obtain accurate optical flow estimates. By the same reasoning, we expect the MRE to be higher but the duty cycle to be lower when the threshold is high. We present this tradeoff in Figures 3-9 to 3-13, where each point labeled *This Work* in the legend represents the median duty cycle and MRE pair across all of the sequences in each dataset for different thresholds.

3.4.5 Impact of Optical Flow Algorithm

To quantify the impact of the TSS algorithm on the overall accuracy of estimated depth maps, we compare our algorithm to a variant that uses the Lucas Kanade

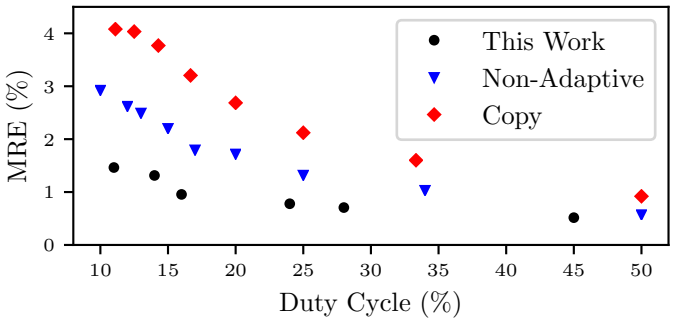


Figure 3-9: **Tradeoff Between Duty Cycle and MRE for TU Munich RGB-D [85]**: We compare our technique (This Work) to the following: Non-Adaptive (Section 3.4.6) and Copy (Section 3.4.7). Because our technique is adaptive, our duty cycles do not align with the competing techniques, which estimate depth at regular intervals.

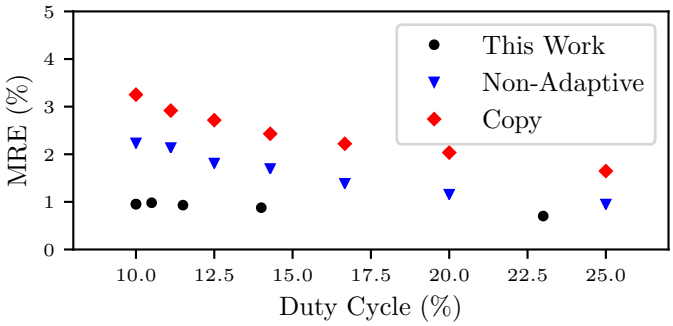


Figure 3-10: **Tradeoff Between Duty Cycle and MRE for NYU Depth V2 [83]**: We compare our technique (This Work) to the following: Non-Adaptive (Section 3.4.6) and Copy (Section 3.4.7). Because our technique is adaptive, our duty cycles do not align with the competing techniques, which estimate depth at regular intervals.

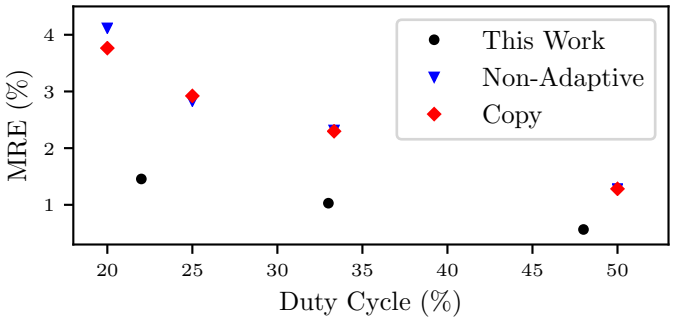


Figure 3-11: **Tradeoff Between Duty Cycle and MRE for Indoor RGB-D [80]**: We compare our technique (This Work) to the following: Non-Adaptive (Section 3.4.6) and Copy (Section 3.4.7). Because our technique is adaptive, our duty cycles do not align with the competing techniques, which estimate depth at regular intervals.

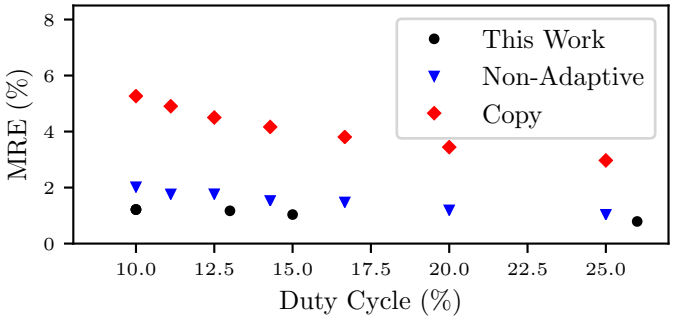


Figure 3-12: **Tradeoff Between Duty Cycle and MRE for CoRBS [90]**: We compare our technique (This Work) to the following: Non-Adaptive (Section 3.4.6) and Copy (Section 3.4.7). Because our technique is adaptive, our duty cycles do not align with the competing techniques, which estimate depth at regular intervals.

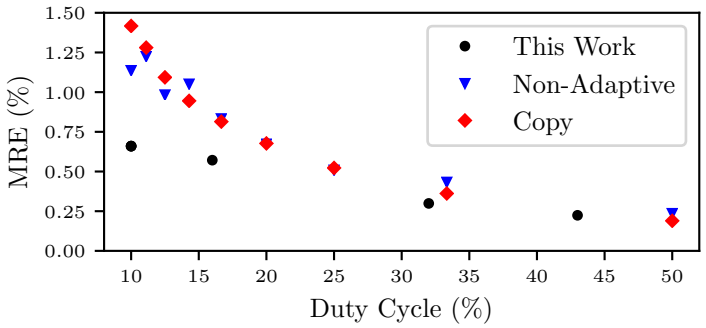


Figure 3-13: **Tradeoff Between Duty Cycle and MRE for ICL-NUIM [24]**: We compare our technique (This Work) to the following: Non-Adaptive (Section 3.4.6) and Copy (Section 3.4.7). Because our technique is adaptive, our duty cycles do not align with the competing techniques, which estimate depth at regular intervals.

algorithm to estimate optical flow. We expect the TSS algorithm to perform worse than the Lucas Kanade algorithm in estimating the optical flow because the TSS algorithm only considers select locations in its search for the best matching block, and to increase the overall MRE of the estimated depth map. In Tables 3.4 to 3.6, we compare this variant (*LK*) to our approach (*This Work*) for the same duty cycle.

Dataset	**This Work**	**LK**	**Non-Adaptive**	**Copy**	**SfM-SIFT**
TU Munich RGB-D	0.96	0.86	1.80	3.20	36.14
NYU Depth V2	0.95	0.82	2.24	3.25	43.03
Indoor RGB-D	1.03	1.49	2.32	2.30	32.29
CoRBS	1.04	1.02	1.54	4.16	38.61
ICL-NUIM	0.67	0.14	1.14	1.42	39.76
Mean	0.93	0.87	1.81	2.87	37.97
Median	0.96	0.86	1.80	3.20	38.61

Table 3.4: **MRE Comparison**: We compare the MRE (%) of our algorithm to variants and competing techniques for approximately the same duty cycle to show that our approach estimates accurate depth maps.

Dataset	**This Work**	**LK**	**Non-Adaptive**	**Copy**	**SfM-SIFT**
TU Munich RGB-D	2.27	1.68	6.26	5.80	83.77
NYU Depth V2	4.04	3.34	5.98	10.04	171.00
Indoor RGB-D	2.11	3.53	5.31	5.41	119.32
CoRBS	1.79	1.66	2.94	9.44	80.52
ICL-NUIM	2.04	0.39	3.21	3.98	126.38
Mean	2.45	2.12	4.74	6.93	116.20
Median	2.11	1.68	5.31	5.80	119.32

Table 3.5: **MAE Comparison**: We compare the MAE (cm) of our algorithm to variants and competing techniques for approximately the same duty cycle to show that our approach estimates accurate depth maps.

From this comparison, we see that our hypothesis is confirmed and that using the Lucas Kanade algorithm in our pipeline reduces the overall median MRE from 0.96% to 0.86%. However, while the Lucas Kanade algorithm reduces the MRE of the estimated depth maps by over 10%, it does not justify the 50% decrease in the estimation frame rate when profiled on the ODROID board. As shown in Table 3.7, its frame rate is 15 FPS, which is intolerable for real time applications.

Dataset	This Work	LK	Non-Adaptive	Copy	SfM-SIFT
TU Munich RGB-D	7.63	7.55	13.83	25.97	104.99
NYU Depth V2	9.01	8.11	14.65	40.25	212.91
Indoor RGB-D	7.54	13.20	14.96	20.04	154.14
CoRBS	8.98	9.82	12.28	34.87	106.66
ICL-NUIM	5.65	3.02	8.26	10.62	158.74
Mean	7.76	8.34	12.80	26.35	147.49
Median	7.63	8.11	13.83	25.97	154.14

Table 3.6: **RMSE Comparison**: We compare the RMSE (cm) of our algorithm to variants and competing techniques for approximately the same duty cycle to show that our approach estimates accurate depth maps.

Algorithm	Frame Rate (FPS)
This Work	30
LK	15
Non-Adaptive [62]	30
Copy [89]	0.83
SfM-SIFT [5]	0.12
SfM-SURF [5]	0.36
SfM-ORB [5]	1.81

Table 3.7: **Algorithm Frame Rate Comparison**: We compare the estimation frame rates our approach and other techniques on the ODROID-XU3 board [26].

3.4.6 Benefit of Adaptive Estimation Due to RANSAC

One key feature of our algorithm is that it adaptively uses the ToF camera when an accurate depth map cannot be estimated. This is necessary because it is not always possible to obtain accurate optical flow estimates. We compare our adaptive scheme to our previous work [62], which predicts depth maps at regular intervals. We apply this approach to the datasets in Section 3.4.2 and plot the duty cycle and MRE pairs, which are denoted as *Non-Adaptive* in Figures 3-9 to 3-13.

In these figures, we see that the adaptive scheme of our approach outperforms [62] across all duty cycles with a negligible increase in complexity. For the same duty cycle, we see in Table 3.4 that our adaptive schemes reduces the median MRE from 1.80% to 0.96%. Furthermore, this result make sense upon inspecting the images in the datasets. Images with rapid motion are blurred and contain large displacements,

making the estimation of accurate optical flow challenging. Our algorithm is optimized to detect these scenarios and uses the ToF camera while estimating depth maps for frames with slower motion.

3.4.7 Comparison to Previous Work

Temporal Depth Map Estimation

We compare our algorithm to a causal variant of [89] as described in Section 3.2.1. This technique estimates depth by copying previous measurements guided by the optical flow. Since our setup requires depth maps to be estimated in real time, we use the optical flow between the current and preceding images to copy the depth from a previous frame. In our experiments, we compute a dense optical flow field using [16]. The estimation of dense optical flow is prohibitively slow on our embedded processor, and this technique, which we denote as *Copy*, runs at 0.83 FPS as shown in Table 3.7. However, we still perform this experiment to quantify the effectiveness of remapping depth. We expect this approach to perform well when the motion between frames is small.

We apply this approach to the datasets and plot the duty cycle and MRE pairs in Figures 3-9 to 3-13. From these figures, we see that our approach outperforms *Copy* across all duty cycles and datasets. This result shows that our dataset contains non-trivial changes in depth that cannot be captured by simply remapping the pixels of a previous depth map. Furthermore, this experiment suggests that the changes in depth can be estimated by our technique.

Structure-from-Motion

We also compare our algorithm to a structure-from-motion (SfM) pipeline that estimates relative depth. Even though SfM estimates relative depth at a sparse set of points, these techniques only use images and can be compelling if it can run in real-time on a low-power embedded platform. We implement an incremental SfM pipeline following standard and state of the art approaches described in [5]. We use

SIFT [55] to localize keypoints, match consecutive keypoints using brute force matching, perform geometric validation using the 8 point algorithm, and triangulate using the direct linear transform method [28]. We apply the SfM pipeline to our setup and estimate the depth using two consecutive images. Across the different datasets, our SfM pipeline estimates the depth at approximately 210 keypoints.

As summarized in Table 3.7, our implementation (*SfM-SIFT*) runs at 0.12 FPS on the ODROID XU-3 board, where most of the time is spent on computing and matching the keypoints. Due to the low frame rate, we also experimented with using SURF [4] (*SfM-SURF*) and ORB [77] (*SfM-ORB*) features instead of SIFT. These variants estimate sparse depth at 0.36 and 1.8 FPS, respectively. While these variants have a higher frame rate than the standard pipeline, they are still far from real time. To quantify the accuracy of the depth estimates obtained using the standard SfM pipeline, we find the scale factor so that the estimated relative depth best matches the ground truth. We summarize the MRE for each dataset in Table 3.4, where we also compare it (*SfM-SIFT*) to our approach and other competing techniques. Because our pipeline uses only two images, the high MRE is expected. We can lower the MRE by incorporating more frames and performing bundle adjustment [28], but this would increase latency and further decrease the estimation frame rate. Due to the high MRE and the low frame rate, we see that SfM is impractical for the scenario we consider.

3.5 Augmented Reality Example

In addition to evaluating the accuracy of the estimated depth maps, we also qualitatively show that they can be used in an actual augmented reality (AR) pipeline that renders 3D models. We insert our depth map estimation algorithm into the pipeline, where the estimated depth maps are used to help localize the camera position and to estimate the surface normals in the scene. For AR, localizing the camera enables rendered objects to remain in fixed locations as the user (e.g., camera) moves. To localize the camera position, our AR pipeline uses ORB-SLAM [59] on the consecutive

image and depth map pairs. We also estimate the surface normals using the estimated depth maps to correctly orient the 3D models. In Figure 3-14, we show select frames from this AR application, where we see that the object is correctly oriented and remains fixed as the camera moves.

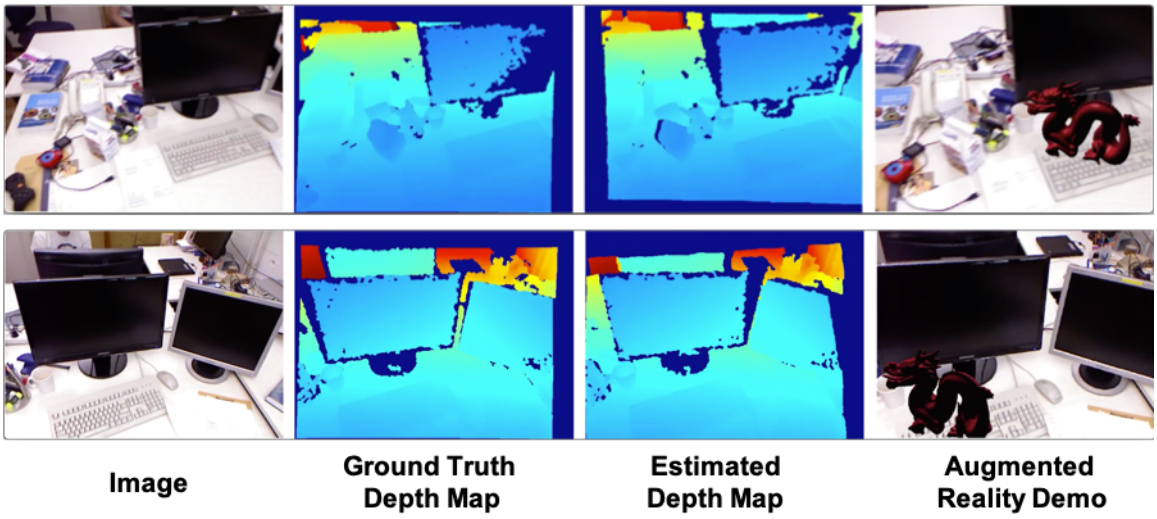


Figure 3-14: **Augmented Reality**: We show that our estimated depth maps can be used in an actual augmented reality pipeline.

3.6 System Power Reduction

To lower the overall power for depth sensing, our strategy is to lower the duty cycle of the ToF camera and estimate depth maps instead. However, this implies that the power required to estimate a new depth map is less than that of using a ToF camera. Here, we measure the power of an implementation of our algorithm on the ODROID XU-3 board and use it to estimate the overall system power of a system that uses our algorithm alongside the ToF camera to obtain depth.

The ODROID XU-3 board has 4 Cortex-A7 CPUs and 4 Cortex-A15 CPUs. To keep the computation power low, we only use the Cortex-A7 cores to estimate the depth maps. This leaves the Cortex-A15 cores available for other mobile applications that use depth maps and further underscores that our implementation, which outputs 640×480 depth maps in real time, is efficient. The resulting implementation consumes

a total of 0.69 W, of which the idle power is 0.19 W. We summarize the power breakdown of our implementation in Table 3.8.

Category		Power (W)
Core	Active	0.63
	Idle	0.16
DRAM	Active	0.06
	Idle	0.03
Total	Active	0.69
	Idle	0.19

Table 3.8: **Power Breakdown**: We measure the power of our implementation on the ODROID-XU3 board [26].

Given the power of our implementation, we now estimate the overall system power of a hybrid system that uses the ToF camera and our algorithm to obtain depth. We define the overall system power, denoted as P_S, as follows:

$$P_S = \frac{ON}{100} \cdot (P_{ToF} + P_I) + (1 - \frac{ON}{100}) \cdot (P_C + P_M) \tag{3.3}$$

where we denote ON as the duty cycle of the ToF camera, P_{ToF} is the power of the ToF camera, P_I is the total idle power, P_C is the active power of the A7 cores, and P_M is the active power of the DRAM. Because we assume that images are routinely collected for other purposes, we ignore its contribution in Eq. (3.3). Based on a survey of commercial ToF cameras (with ranges up to 4 meters), we also assume that P_{ToF} ranges from 1 to 5 W [2] [12].

Taking the duty cycle to be 15% and using the measurements in Table 3.8, we plot the power of the hybrid system in Figure 3-15. For the datasets in Table 3.3, this translates to a median power reduction of 23%-73% compared to just using the ToF camera while producing depth maps with a median MRE of 0.96%.

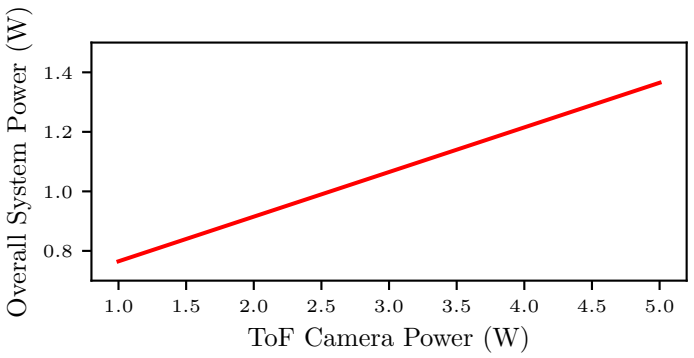


Figure 3-15: **Overall System Power**: We estimate the power of a system that uses our algorithm to estimate depth alongside the ToF camera. For commercial ToF cameras, our algorithm can reduces the overall system power by 23%-73%.

3.7 Infilling Depth Maps

In the previous section, we describe how we use our algorithm to estimate new depth maps *temporally* to lower the power for ToF imaging. Here, we show that our algorithm can also be used to estimate depth *spatially* to infill missing depth values. This means that our algorithm can be used to address two deficiencies of ToF imaging, namely when the sensor goes out of range and when the sensor saturates (e.g., when the reflected light overloads the image sensor). We consider both cases and show how we can, in effect, extend the range of a ToF camera without increasing the power of its illumination source and overcome saturation.

We first consider the scenario where the ToF camera goes out of range by acquiring images and depth maps of a scene shown in the first image of Figure 3-16. We use the Pico Zense DCAM710 RGB-D sensor [68], which contains a ToF and digital camera that outputs 640×480 depth maps and 1080×1920 RGB images, respectively. We expect that as we move the sensor away from the objects in the scene, we will not be able to measure depth for every object. We show an instance of this in the second image of Figure 3-16, where the ToF camera goes out of range and the depth for the box is unknown. To infill the depth values for the box, we use a previously measured image and depth map pair, where depth is available for the box, and the current image to estimate a new depth map, which is shown in the last image of

Figure 3-16. One limitation of our approach is that we can only infill regions where we have previous depth, and in this case, we cannot estimate depth for the wall. To evaluate the accuracy of our depth map, we compute the mean relative error for the overlapping pixels between the measured and estimated depth maps. Because the scene is rigid, which means that the relative distance between the box and chair does not change, we expect that this mean relative error is also representative of what we would obtain if the depth for the box is available in the measured depth map. In this example, we achieve a mean relative error of 0.87%.

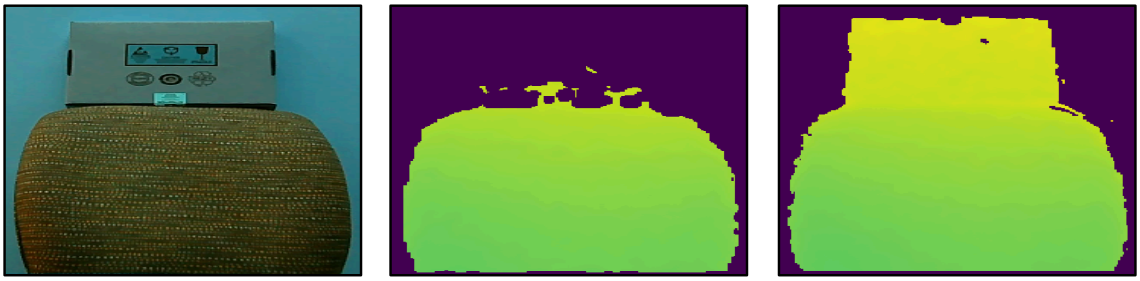


Figure 3-16: **Out of Range**: We estimate the depth for objects that exceed the ToF camera's range using our algorithm. The purple regions cannot be sensed by the ToF camera.

We also consider the scenario where a ToF camera becomes saturated by acquiring images and depth maps of a scene, shown in the first image of Figure 3-17, as we move the ToF camera closer to the book. As shown in the second image of Figure 3-17, the sensor saturates and depth is not available in the center of the book. By using a previous image and depth map pair, we are able to overcome this deficiency and estimate depth in this region, achieving a mean relative error of 0.6% for the overlapping pixels.

3.8 Summary

In this chapter, we present an algorithm to estimate causal depth maps for rigid objects and scenes using concurrently collected images and previously measured depth. We use this approach to reduce the power of ToF imaging. Instead of using the ToF

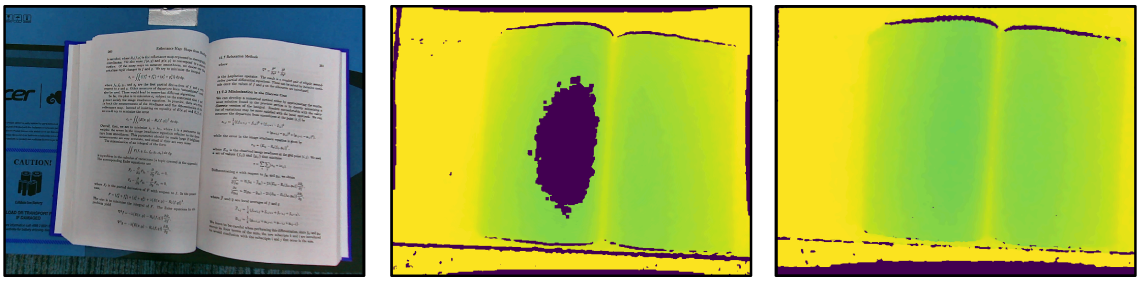


Figure 3-17: **Saturation**: We estimate the depth for pixels that are saturated using our algorithm. The purple regions cannot be sensed by the ToF camera.

camera continuously to acquire depth, we estimate depth maps using our technique and only use the ToF camera when an accurate depth map cannot be estimated. To ensure that the overall power for depth sensing is reduced, we design our algorithm to run efficiently on a low power embedded platform by obtaining new depth maps using the pose, which can be estimated using *sparse* operations. The resulting implementation produces 640×480 depth maps in real time, or 30 frames per second. We evaluated our approach on several RGB-D datasets, where our technique produces depth maps with a mean relative error of 0.96% and lowers the usage of the ToF camera by 85%. When used with commercial ToF cameras, our algorithm can reduce the total power for depth sensing by up to 73%.

Chapter 4

Depth Map Estimation for Non-Rigid Objects and Dynamic Scenes

4.1 Introduction

While the rigidity assumption is made for many applications, in reality, many still operate in non-rigid and dynamic environments. In this chapter, we show how we can use concurrently collected RGB images to reduce the usage of a time-of-flight (ToF) camera and estimate new depth maps for *non-rigid objects* and *dynamic scenes* as shown in Figure 4-1. We define a non-rigid object as an object that deforms such that the pair-wise distance between any two points can change from frame to frame while remaining intact. An example of a non-rigid object is a bending sheet of paper. We also consider dynamic scenes, which we define as a collection of rigid and non-rigid objects that have independent motions. We show examples of these different scenarios in Figure 4-2.

To estimate the depth map for both non-rigid objects and dynamic scenes, we take an approach similar to that in Chapter 3 by first estimating the 3D motion in the scene and then using it to update a previously measured depth map. We estimate the 3D motion for these objects and scenes by assuming that they are *locally rigid*, which means that the points in a small spatial neighborhood, or region, behave like a rigid object. For the non-rigid objects, neighboring regions have similar rigid motions,

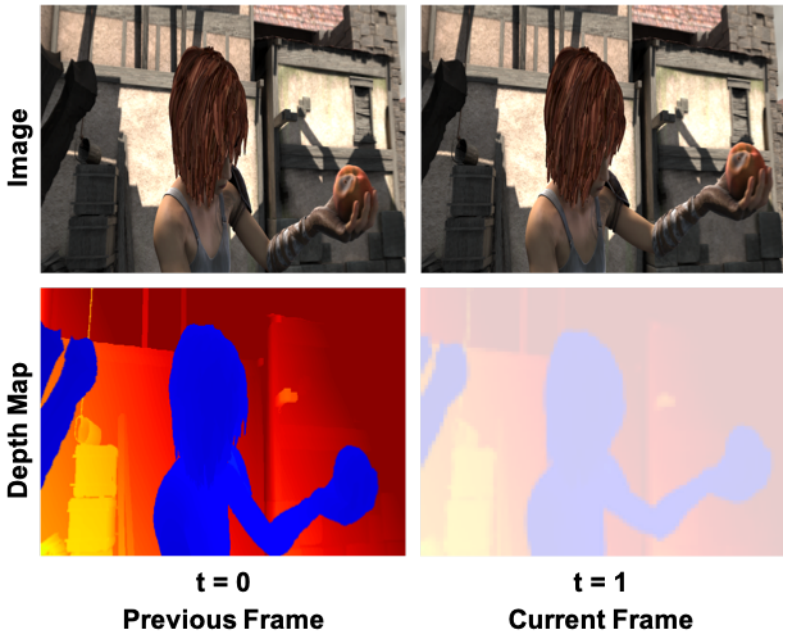


Figure 4-1: **Depth Estimation Setup**: The inputs to our algorithm are two consecutive and concurrently collected images and a previous depth map. The algorithm then estimates the current depth map. Because the previous depth map can either be measured or estimated, our technique can be used to sequentially estimate depth to further reduce the usage of the ToF camera.

whereas for dynamic scenes, these regions can have substantially different rigid motions. These differences will result in different depth map estimation algorithms, but the assumption of local rigidity allows both algorithms to estimate the 3D motion in each scenario without explicit rigid motion segmentation. This allows us to efficiently estimate accurate depth maps and reduce the sensor power of the ToF camera. The contributions of this chapter are as follows:

- For both non-rigid objects and dynamic scenes, we show that we can estimate their 3D motion using the optical flow and depth at a *sparse set of pixels*. This reduces the latency in which we obtain depth maps.

- For non-rigid objects, we show that we can accurately estimate depth maps by subdividing the pixels in the previous frame into overlapping rigid segments and constraining the pixels in the overlapping regions to have the same motion. We show that we can formulate this as a *sparse linear system*, which can be

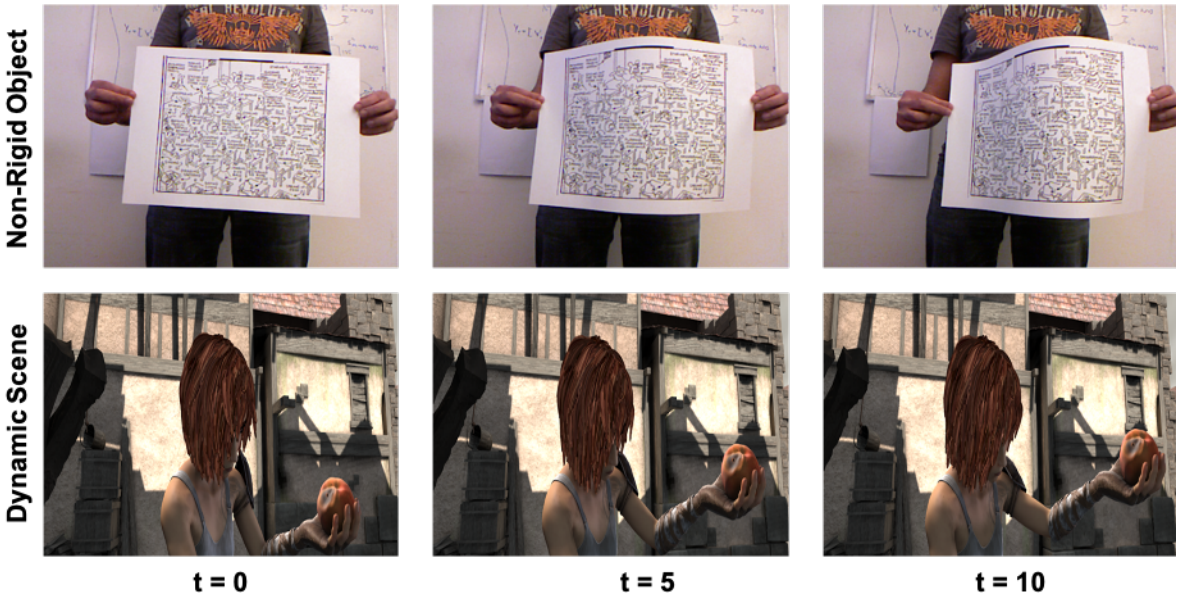


Figure 4-2: **Non-Rigid Objects and Dynamic Scenes**: We show frames from two sequences that depict either a non-rigid object or a dynamic scene. In the non-rigid object example, the sheet of paper is bending and remains intact. In the dynamic scene, the character, her hand, and the background are all moving independently of each other.

efficiently solved and increase the throughput of our approach.

- For dynamic scenes, we show that the rigid motions in the scene can be *clustered* and assigned to the pixels of the previous depth map by using the *photometric error* between the current image and one obtained by reprojecting the previous image. Compared to rigid motion segmentation, computing the photometric error is efficient as its complexity is linear with respect to the number of pixels in the image. This increases the throughput at which we estimate accurate depth maps.

The resulting algorithms estimate dense depth maps in up to real-time using the CPUs of a laptop computer (i5-5257U dual core) and outperform similar approaches in the literature in terms of both accuracy and latency. While our algorithms are not as efficient as the technique proposed in Chapter 3, these algorithms can support a larger variety of scenes and applications.

This chapter is based on our work in [63,65] and is organized as follows. In Section

4.2, we describe similar techniques that estimate depth maps using consecutive images for non-rigid objects and dynamic scenes and highlight why they are insufficient for our purpose. The remainder of this chapter is divided into two parts. In Part I (Sections 4.3 to 4.4), we focus on non-rigid objects. In Section 4.3, we first describe how we estimate the depth maps for non-rigid objects using the data from consecutive RGB images and a previous depth map. We then evaluate our approach using both synthetic and real datasets in Section 4.4. In Part II (Sections 4.5 to 4.6), we focus on dynamic scenes. In Section 4.5, we describe how we estimate depth maps for dynamic scenes using the data from consecutive RGB images and a previous depth map without computing dense optical flow or rigid motion segmentation. This is followed by an evaluation of our algorithm in Section 4.6, where we also compare it to similar techniques. Finally, we summarize and conclude this chapter in Section 4.7.

4.2 Background

Many approaches have been proposed to estimate depth for non-rigid objects and dynamic scenes for a variety of different applications. Given its breadth, we only summarize techniques that estimate depth with a similar setup or those that only use consecutive RGB images. However, many of these approaches are insufficient for the applications that we consider because they estimate depth maps with either high latency or high computational complexity. We list these methods in Table 4.1, and we highlight the different features that pertain to latency and computational complexity. For latency, we note whether the technique is non-causal or if it is a multi-frame approach (requiring more than 2 consecutive frames to estimate a depth map). For computational complexity, we note whether the technique requires dense optical flow (denoted as Dense Flow in the table), segmentation, or is a deep neural network (denoted as DNN) based approach. These features are all computationally expensive [19, 92].

4.2.1 Depth Transfer Methods (DTM)

Similar to our approach, the depth transfer methods estimate new depth maps using previously measured ones [11, 43, 50, 89, 95]. However, instead of estimating the 3D motion in the scene, these approaches instead estimate the *dense* optical flow between the current image and one that corresponds to a previous depth map and use the optical flow to warp the previous depth map to obtain a new one.

The authors of [11,50,89,95] use this framework to equalize the frame rate between recorded image and depth video. They estimate new depth maps using frames where both images and depth maps are available, and these techniques have the advantage that they have data from both the preceding and *future* frames. While these techniques are effective for increasing the frame rates of depth videos, they are not causal. While Wang *et al.* [89] and Li *et al.* [50] can be adapted to causally estimate depth using only two frames, these approaches do not account for changes in depth since the preceding depth maps are simply warped. This is sufficient for small changes in depth or in-plane motion, but it cannot be generalized to all non-rigid objects and dynamic scenes.

Karsch *et al.* [43] similarly warps a previously measured depth map, but differs in that it uses depth maps taken from a training set of image and depth map pairs of similar scenes. One benefit of this approach is that it can support monocular depth estimation. To estimate the depth maps for consecutive images, these authors improve accuracy by using motion cues to estimate temporally consistent depth maps. This method fails when the training set does not contain image and depth map pairs of similar scenes. As our approaches do not rely on a training set, they do not suffer from this issue.

4.2.2 Non-Rigid Structure-from-Motion (NRSFM)

Non-rigid structure-from-motion techniques estimate relative depth using only images by exploiting statistical and physical heuristics [42]. Here, we focus on the methods that estimate dense depth maps in dynamic scenes, namely [46, 47, 73, 76, 94]. Like

our approach, these methods are all causal and with the exception of [76,94] estimate depth using only two consecutive frames.

However, unlike our approach, these techniques have high complexity. The authors of [73, 76, 94] estimate depth by first segmenting the pixels into rigid regions. Zhang *et al.* [94] directly segments the pixels whereas Roussos *et al.* [76] and Ranftl *et al.* [73] first compute a dense optical flow field and then segment it to initialize their algorithms. This comes with a high computational cost and lowers the throughput at which depth can be estimated, a detriment for many applications. However, this rigid motion segmentation is necessary because estimating depth is underdetermined without geometric assumptions like rigidity. Noting the challenge of rigid motion segmentation, other approaches use simpler partitions [46,47]. Kumar *et al.* [46] assumes that dynamic scenes are locally rigid and partitions the scene into rigid superpixels. This technique then computes the dense optical flow and uses it to estimate the depth within each superpixel. Our approaches are similar to these techniques in that they model both non-rigid objects and dynamic scenes using rigid motions, but they avoid both dense optical flow and prior segmentation. This is because our techniques leverage the previous depth map to both estimate the rigid motions and assign them to the pixels of the previous depth map. As we will show, this enables efficient depth map estimation.

4.2.3 Neural Networks for Depth Estimation (NNFDE)

In addition to the previous approaches, many deep neural networks have recently been proposed to estimate depth using monocular images. We focus on techniques that use consecutive images to estimate depth in dynamic scenes [9, 23, 51]. In addition to consecutive images, some of these methods require prior segmentation as an input before depth can be estimated. Casser *et al.* [9] and Li *et al.* [51] require object-level segmentation masks whereas Gordon *et al.* [23] requires bounding boxes of possibly mobile objects. Li *et al.* [51] even requires an initial depth map obtained using structure-from-motion techniques, underscoring the inherent difficulty of estimating depth for dynamic scenes. While these techniques are promising, they are

computationally complex, and their performance is limited by the diversity of their training set. Unlike these approaches, our techniques estimate depth by exploiting physical heuristics and do not require a training set.

Across a variety of different approaches that estimate depth in dynamic scenes, we see that many methods require a dense optical flow field or prior segmentation. As summarized in Table 4.1, none of these approaches satisfy our requirements. In this chapter, we show how we can avoid these operations for our problem setup to efficiently estimate depth for both non-rigid objects and dynamic scenes.

4.3 Part I: Depth Map Estimation of Non-Rigid Objects

In this section, we describe how we reduce the usage of the ToF camera and estimate new depth maps for non-rigid objects. As summarized in Figure 4-3, our algorithm takes as input two consecutive images and a previous depth map. To estimate the non-rigid deformation and depth, our algorithm assumes that non-rigid objects are composed of locally rigid segments. As such, our technique first partitions the pixels into rigid regions. We then use the optical flow to estimate the 3D motion and depth of each region. We describe this next.

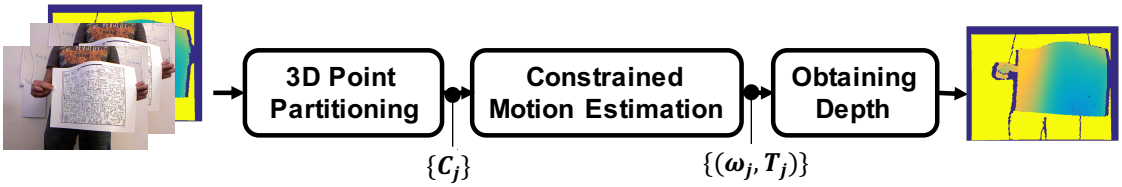


Figure 4-3: **Non-rigid Depth Map Estimation Pipeline:** Our algorithm takes as input consecutive images and previous depth measurements. Nearby pixels are partitioned into regions that have the same rigid motion. Because regions can overlap, we solve a constrained optimization problem to estimate the new 3D position of each point and then obtain depth.

Method	Category	Prior Depth?	High Latency		High Complexity		
			Non-Causal	Multi-Frame	Dense Flow	Segmentation	DNN
This Work		✓					
Choi *et al.* [11]	DTM	✓	✓	✓	✓		
Wang *et al.* [89]	DTM	✓			✓		
Li *et al.* [50]	DTM	✓			✓		
Karsch *et al.* [43]	DTM	✓			✓		
Zhang *et al.* [95]	DTM	✓	✓	✓	✓		
Zhang *et al.* [94]	NRSFM			✓		✓	
Roussos *et al.* [76]	NRSFM			✓	✓	✓	
Ranftl *et al.* [73]	NRSFM				✓	✓	
Kumar *et al.* [46]	NRSFM				✓		
Kumar *et al.* [47]	NRSFM				✓		
Casser *et al.* [9]	NNFDE			✓		✓	✓
Gordon *et al.* [23]	NNFDE						✓
Li *et al.* [51]	NNFDE				✓	✓	✓

Table 4.1: **Related Works Comparison**: We summarize the techniques in Section 4.2 that estimate dense depth maps for non-rigid objects and dynamic scenes. For each technique, we use a checkmark to highlight features that are related to its latency and computational complexity. As our goal is to estimate depth maps with low latency and high throughput, we see that these previous techniques are insufficient.

4.3.1 3D Point Partitioning

Our approach assumes that nearby points undergo the same rigid motion. As such, we first group these points together using their 3D coordinates. We first compute the 3D coordinate of the pixels in the previous depth map using Eq. (2.2). We then assign each 3D point to regions that are centered on predefined centers. For example, the region centered on the j^{th} predefined point is the following set of points:

$$C_j = \{\boldsymbol{P_i} : ||\boldsymbol{P_i} - \boldsymbol{P_j}||_2 < \epsilon \quad i = 1, 2, \ldots, N\} \tag{4.1}$$

where we denote $\boldsymbol{P_i}$ as the 3D coordinate of the i^{th} point, ϵ as the radius of each region, and N as the total number of 3D points that are partitioned. Because neighboring pixels have similar motion and depth, we only partition the pixels on a uniformly spaced grid to reduce computation. We then distribute the region centers uniformly across this grid. In our experiments (Section 4.4), we find that a 10×15 grid with regions centered on every tenth pixel (with $\epsilon = 70$) offered the best tradeoff between the accuracy of the estimated depth maps and the latency in which they are obtained.

Because the points in each region are rigid, they must satisfy Eq. (2.17) and Eq. (2.18), which we duplicate here in matrix form:

$$\begin{pmatrix} \frac{f}{Z_i} & 0 & -\frac{x_i - x_c}{Z_i} & -\frac{(x_i - x_c)(y_i - y_c)}{f} & f + \frac{(x_i' - x_c)^2}{f} & -(y_i - y_c) \\ 0 & \frac{f}{Z_i} & -\frac{y_i - y_c}{Z_i} & -f - \frac{(y_i - y_c)^2}{f} & \frac{(y_i - y_c)(x_i - x_c)}{f} & (x_i - x_c) \end{pmatrix} \begin{pmatrix} \boldsymbol{T} \\ \boldsymbol{\omega} \end{pmatrix} = \begin{pmatrix} \dot{x}_i \\ \dot{y}_i \end{pmatrix} \tag{4.2}$$

where (x_i, y_i) is the 2D pixel coordinate of $\boldsymbol{P_i}$, $(\dot{x}_i, \dot{y}_i)$ is its optical flow, f is the focal length, and (x_c, y_c) is the principal point. Here, we use angular and translational velocity to represent the rigid motion within each region, which we denote as $\boldsymbol{T}$ and $\boldsymbol{\omega}$, respectively.

To determine the points that have the same rigid motion, we first compute the optical flow using the Lucas Kanade algorithm [56] and then solve the linear system that results from Eq. (4.2) with RANSAC to determine the inlier set, or the set of points that have residual errors within a threshold ρ. For each region, we only retain the points in this inlier set. In our experiments, we empirically find that $\rho = 0.5$

results in accurate depth maps, and we further discuss the impact of RANSAC in Section 4.4.4.

4.3.2 Constrained Motion Estimation

Once the points are partitioned, we want to estimate the rigid motion for each region to obtain the new 3D positions. However, given the placement of the region centers, some points belong to multiple regions. We need to ensure that the new position of each point is consistent across the regions it belongs to. For these points, we have the following consistency constraint:

$$\boldsymbol{\omega_k} \times \boldsymbol{P_i} + \boldsymbol{T_k} = \boldsymbol{\omega_l} \times \boldsymbol{P_i} + \boldsymbol{T_l} \tag{4.3}$$

where $\boldsymbol{P_i} \in C_k \cap C_l$, which means that the i^{th} point belongs to both the k^{th} and l^{th} region. Without loss of generality, we also denote $\boldsymbol{\omega_j}$ and $\boldsymbol{T_j}$ as the angular and translational velocity of the j^{th} region, respectively. We rewrite Eq. (4.3) in matrix form, which is convenient for our final formulation.

$$\begin{pmatrix} \boldsymbol{I} & -[\boldsymbol{P_i}]_\times & -\boldsymbol{I} & [\boldsymbol{P_i}]_\times \end{pmatrix} \begin{pmatrix} \boldsymbol{T_k} \\ \boldsymbol{\omega_k} \\ \boldsymbol{T_l} \\ \boldsymbol{\omega_l} \end{pmatrix} = \boldsymbol{0} \tag{4.4}$$

where we denote $[\boldsymbol{P_i}]_\times$ as the skew-symmetric matrix such that $[\boldsymbol{P_i}]_\times \boldsymbol{P_j} = \boldsymbol{P_i} \times \boldsymbol{P_j}$ and $\boldsymbol{I}$ as the identity matrix.

To estimate the motion, we then combine the rigidity constraints in Eq. (4.2) with the consistency constraints above to formulate an optimization problem to estimate the motion within each region:

$$\begin{aligned} \min_{\boldsymbol{\beta}} \quad & \frac{1}{2}||\boldsymbol{A\beta} - \boldsymbol{\Delta}||_2^2 \\ \text{subject to:} \quad & \boldsymbol{D\beta} = \boldsymbol{0} \end{aligned} \tag{4.5}$$

where $\boldsymbol{\beta}$ is the concatenation of the rigid motions in all of the regions, $\boldsymbol{A}$ and $\boldsymbol{\Delta}$ contain the rigidity constraints in Eq. (4.2) for each point, and $\boldsymbol{D}$ contain the consistency constraints in Eq. (4.4) for the points that belong to multiple regions.

It should be noted that the constraint $\boldsymbol{D\beta} = \mathbf{0}$ makes neighboring regions with overlapping points have *similar* rigid motion as described in Section 4.1. When there are more than 3 overlapping pixels, this constraint merges the regions. While this may seem like a limitation, this is by design because intuitively, regions with significant overlap should have the same rigid motion. In our experiments, we empirically chose the partitioning parameters to avoid overlapping regions unless the regions have similar rigid motions.

The solution, $\boldsymbol{\beta}$, to Eq. (4.5) can be found by solving the following *unconstrained* linear system:

$$\begin{pmatrix} \boldsymbol{A^T A} & \boldsymbol{D^T} \\ \boldsymbol{D} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A^T \Delta} \\ \mathbf{0} \end{pmatrix} \tag{4.6}$$

where $\boldsymbol{\lambda}$ is a vector of the Lagrange multipliers that enforce the equality constraints. The matrix in Eq. (4.6) is *sparse*, where $\boldsymbol{A^T A}$ is block-wise diagonal and every row of $\boldsymbol{D}$ contains at most six elements. Therefore, this system can be efficiently solved. For Eq. (4.6) to have a unique solution, we note that the columns of $\boldsymbol{D^T}$ and $\boldsymbol{(A^T, D^T)^T}$ must all be linearly independent, and we can select them using QR factorization.

4.3.3 Obtaining Depth

Once the motion within each region is estimated, we can obtain the new 3D position for the point $\boldsymbol{P_i}$ in the k^{th} region as follows:

$$\boldsymbol{P_i'} = \boldsymbol{P_i} + \boldsymbol{\omega_k} \times \boldsymbol{P_i} + \boldsymbol{T_k} \tag{4.7}$$

To obtain a depth map, we project the depth of each point using the camera intrinsics. Because we estimate motion on a subsampled grid, the resulting depth map is sparse, but we can obtain a dense depth map using linear interpolation.

4.4 Part I: Evaluation for Non-Rigid Objects

4.4.1 Methodology

We evaluate our algorithm on both synthetic and real data that have substantial changes in depth from frame to frame. We synthesize 640×480 planar images bending smoothly (*syn_bend*) and being sharply folded in the middle (*syn_crease*). We also test our algorithm on the RGB-D sequences *kinect_paper* and *kinect_tshirt* from [86]. We crop out the object undergoing the non-rigid deformation to test our algorithm. We assume that we only have depth measurements in the first frame and estimate it until the fourth frame. We quantify the accuracy of our estimated depth maps using the percent mean relative error (MRE) defined as in Eq. (1.10).

4.4.2 Implementation

We implement our algorithm on a laptop with an i5-5257U CPU and an embedded platform [26] with an Exynos 5422 processor. Our laptop implementation can estimate a dense (640×480) and sparse depth map in approximately 17 and 50 frames per second (FPS), respectively, and the bottleneck is linear interpolation. When using only the Cortex-A7 cores on the embedded platform, which consumes 352 mW (Idle Power: 178 mW), our algorithm obtains dense and sparse depth estimates in 3 and 11 FPS, respectively. In contrast, non-rigid structure-from-motion approaches like Kumar *et al.* [46] require minutes to obtain depth on a desktop computer with an i7 processor.

4.4.3 Results

We present the MRE in percentage form for each sequence and frame in Table 4.2. The average MRE across all sequences and frames is 0.37%. We also show an example of a 3D reconstruction of a frame from the *kinect_paper* sequence using our estimated depth map in Figure 4-4.

	Frame Number			
Sequence	**2**	**3**	**4**	**Mean**
kinect_paper	0.19	0.43	0.23	0.28
kinect_tshirt	0.35	0.52	1.16	0.68
syn_bend	0.27	0.25	0.24	0.26
syn_crease	0.27	0.27	0.27	0.27
Mean - Real	0.27	0.48	0.69	0.48
Mean	0.27	0.37	0.47	0.37

Table 4.2: **Non-Rigid Depth Map Results:** We present the percent MRE for each sequence and frame number for both real (*kinect_paper* and *kinect_tshirt*) and synthetic sequences (*syn_bend* and *syn_crease*).

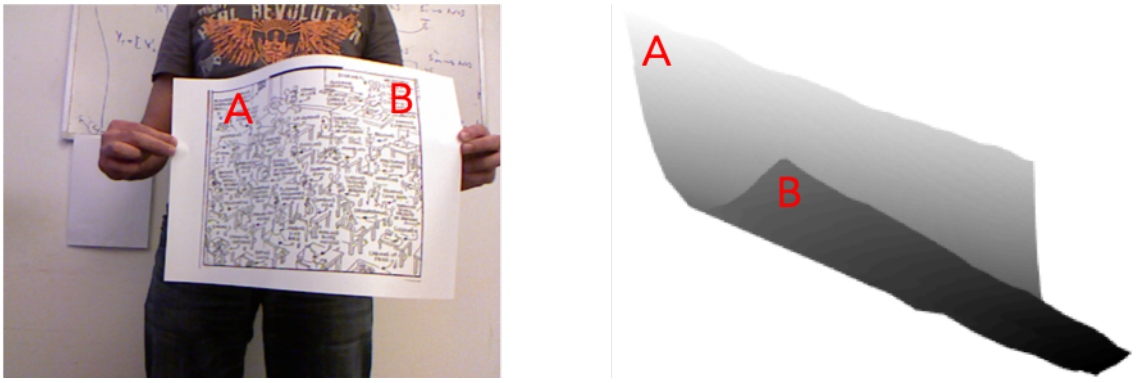


Figure 4-4: **Reconstruction of *kinect_paper*:** We present the 3D reconstruction of the *kinect_paper* using the depth map estimated by our approach. We rotated the paper to show its contours between points A and B.

4.4.4 Discussion

Algorithm Outperforms Non-Rigid Structure-from-Motion Approaches

We compare our algorithm to an adapted approach that first uses techniques like Kumar *et al.* [46] to obtain depth to scale, and then use the previous depth measurements to estimate the unknown scale factor. For approaches that perform monocular depth estimation, this same procedure is followed to evaluate the accuracy of their reconstruction. Kumar *et al.* [46] also uses this procedure to benchmark the performance of their algorithm on sequences from [86].

As summarized in Table 4.3, Kumar *et al.* [46] report a MRE of 4.76% for *kinect_paper* and a MRE of 4.80 % for *kinect_tshirt*. In contrast, our algorithm

Sequence	This Work	Kumar *et al.* [46]
kinect_paper	0.28	4.76
kinect_tshirt	0.68	4.80

Table 4.3: **Non-rigid Structure-from-Motion Comparison:** We compare the percent MRE of our approach to techniques benchmarked in Kumar *et al.* [46].

achieves a lower MRE of 0.28% and 0.68% for these respective sequences. This suggests that integrating previous depth measurements directly into the depth estimation process not only simplifies our algorithm but also improves its accuracy.

Impact of RANSAC in 3D Point Partitioning on MRE

In our experiments, we find that using RANSAC to refine our regions lowers the MRE by up to 25%. Because the MRE is an average statistic computed over all the pixels, this metric alone does not reflect how our refinement step preserves the underlying structure. To show that the additional refinement step preserves the underlying structure of the depth map, we test our algorithm on *syn_crease* and compare the 3D reconstructions in Figure 4-5. We see that without this refinement step, the presence of noisy optical flow estimates and our selection of the point partitioning radius results in a curved plane instead of a sharply folded sheet. This failure mode makes sense because nearby points with different motions are partitioned together, and RANSAC mitigates this.

Figure 4-5: **Comparing Reconstructed Shape:** Using RANSAC to refine our point partition preserves the underlying shape.

4.4.5 Limitations of Our Approach

Our algorithm was designed with the assumption that the optical flow can be accurately estimated. A natural consequence of this is that this technique will fail when the images are textureless. Moreover, even when the images are textured, our partitioning scheme assumes that the texture is well distributed across the non-rigid object. This makes our algorithm sensitive to parameters like ϵ and ρ as previously defined. In the next sections, we describe how we overcome these limitations and generalize this approach to not only estimate depth for non-rigid objects but also dynamic scenes.

4.5 Part II: Depth Map Estimation for Dynamic Scenes

We now describe how we reduce the usage of the ToF camera and estimate new depth maps for dynamic scenes using concurrently collected RGB images. Here, we assume that dynamic scenes contain a collection of rigid and non-rigid objects that have *independent* motions. To estimate these motions, we assume that the scene is locally rigid. However, unlike our algorithm for non-rigid objects, we will not constrain the neighboring rigid regions to have similar rigid motions. The pipeline of our algorithm is depicted in Figure 4-6 and is composed of two major stages: one that estimates the independent rigid motions in the scene (Section 4.5.1) and another that assigns them to obtain a new depth map (Section 4.5.2).

4.5.1 Independent Rigid Motion Estimation

As we previously stated, our technique assumes that the motion in dynamic scenes can be approximated using independent rigid motions. In order to estimate it, we invert an image formation model that relates the 3D motion in the scene to its 2D displacement across the consecutive images. In this section, we describe how we accomplish this.

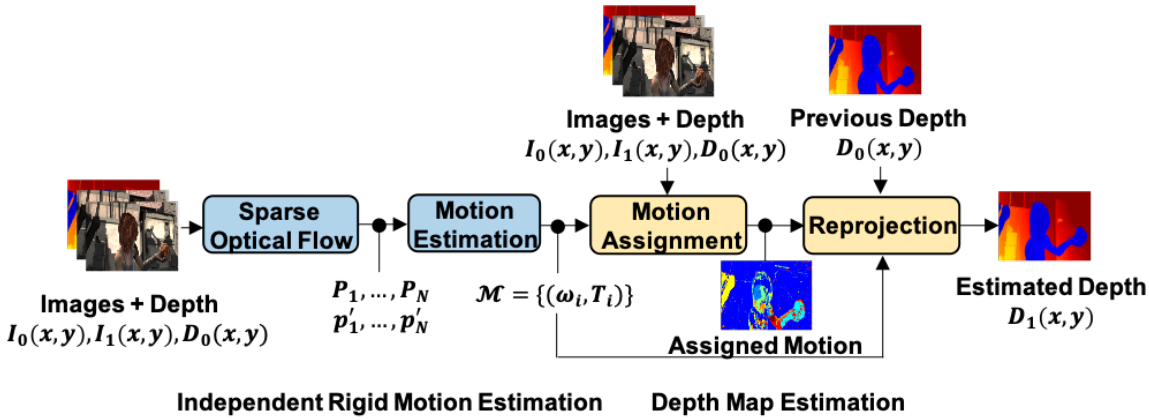


Figure 4-6: **Depth Map Estimation of Dynamic Scene Pipeline**: Our algorithm takes as input two consecutive images and a previous depth map. It uses these inputs to first estimate the independent and rigid motions in the scene using the sparse optical flow computed from the images and its corresponding depth. Our technique then obtains a depth map by assigning the estimated rigid motions to the appropriate pixels of the previous depth map, guided by the photometric error obtained by reprojecting the images.

Sparse Optical Flow

Unlike the approaches described in Section 4.2, our technique estimates the 3D scene motion using the optical flow at a sparse set of pixels. To do so, we first detect corners in the previous image by using the FAST corner detector [75] and then estimate the optical flow at these pixels using the Lucas Kanade algorithm [56]. Compared to our approach for non-rigid objects, which computed the optical flow on a uniformly spaced grid, this is more computationally expensive. However, it is necessary because the independently moving objects in a dynamic scene are positioned throughout the image, and a uniform grid may undersample the pixels that correspond to those objects. Without localizing these pixels, this means that the optical flow and the motion of each object may not be accurately estimated.

If the optical flow for the i^{th} pixel in the previous frame, or $\boldsymbol{p_i}$, is known, then its correspondence in the current frame, $\boldsymbol{p'_i}$, can be trivially found. Furthermore, for every pixel detected by the FAST corner detector, we also compute its 3D coordinate, $\boldsymbol{P_i}$, by using the previous depth map, $D_0(x, y)$, using Eq. (2.2).

Motion Estimation

With the optical flow estimated, we now describe how the independent rigid motions are obtained. In the simplest case, there is a single rigid motion, and we describe how to estimate it in Section 2.2.

To estimate the multiple and independent rigid motions in the scene, we sequentially estimate them individually using RANSAC as previously described. In addition to obtaining the rigid motion, RANSAC also determines the inlier set, which is defined as:

$$\mathcal{I} = \left\{ i : \left|\left| \pi\left(\boldsymbol{P_i'}\right) - \boldsymbol{p_i}' \right|\right|_2^2 \leq \epsilon \right\} \tag{4.8}$$

where ϵ is a threshold based on the projection error.

In our approach, we adapt RANSAC to estimate $\boldsymbol{\omega}$ and $\boldsymbol{T}$ that *maximizes* the size of the inlier set. We then remove the pixels in the inlier set from further consideration and repeat this process to greedily estimate the rigid motions as shown in Figure 4-7. This is done to increase the diversity of the estimated rigid motions to best represent the dynamic motion in the scene and to reduce the complexity of the motion assignment process, described in the next section. This process is repeated until the size of the inlier set falls below a minimum size, N_{min}. As a result, our approach does not need the number of rigid motions to be specified. The output of this algorithm is the set of estimated rigid motions, which we denote as $\mathcal{M} = \{(\boldsymbol{\omega_i}, \boldsymbol{T_i})\}$. We summarize our approach in Algorithm 3.

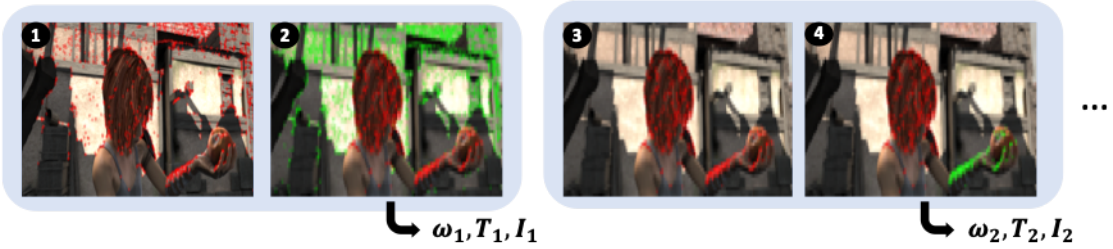


Figure 4-7: **Sequential Rigid Motion Estimation**: We depict how the estimated rigid motions are obtained. In the first image, we highlight the pixels (shown in red) where $\boldsymbol{P_i}$ and $\boldsymbol{p_i}'$ are known. We use RANSAC to estimate the rigid motion with the largest inlier set, shown in green in the second image, and remove these pixels from further consideration. This process is repeated, as shown in the third and fourth images, until the size of the inlier set falls below a threshold.

Algorithm 3 Independent Rigid Motion Estimation

input: $I_0(x, y)$, $I_1(x, y)$, and $D_0(x, y)$; $N_{\min}$, ϵ
output: $\mathcal{M} = \{(\boldsymbol{\omega_i}, \boldsymbol{T_i})\}$
$\mathcal{P} \leftarrow \{(x_i, y_i)\}$ using FAST corner detector on I_0
$\mathcal{P}' \leftarrow \{(x'_i, y'_i)\}$ using Lucas Kanade on $\mathcal{P}, I_0$ and I_1
$\mathcal{Z} \leftarrow \{D_0(x_i, y_i)$ for $(x_i, y_i) \in \mathcal{P}\}$
$\mathcal{M} \leftarrow \{\}$
repeat
    $\boldsymbol{\omega_i}, \boldsymbol{T_i}, \mathcal{I}_i \leftarrow$ Estimate rigid motion with largest inlier set using RANSAC
    if $|\mathcal{I}_i| > N_{\min}$ **then**
        Remove inlier values from $\mathcal{P}, \mathcal{P}', \mathcal{Z}$
        $\mathcal{M} \leftarrow \mathcal{M} \cup (\boldsymbol{\omega_i}, \boldsymbol{T_i})$
    end if
until $|\mathcal{I}_i| < N_{min}$ or $\mathcal{P}$ is empty

4.5.2 Depth Map Estimation

Once the rigid motions are estimated, we obtain a new depth map by using these estimated rigid motions to reproject the previous depth map. To do so, we obtain the 3D position of each pixel, apply the appropriate rigid motion, and project its updated depth. In this section, we describe how we determine which estimated rigid motion to use in order to obtain a new depth map.

Motion Assignment

To determine the motion assignment, we exploit the fact that in the previous frame, both the image and the depth map are spatially aligned. This allows us to reproject the previous image. This is important because if a set of pixels move with a certain rigid motion, then its reprojection must coincide with its corresponding pixels in the next frame. Consequently, the pixel-wise difference between these reprojected pixels and its correspondences in the next frame, or the photometric error, must have a low magnitude. Our approach uses this insight to assign the estimated rigid motions to the appropriate pixels of the previous depth map.

We begin by reprojecting the previous image, or $I_0(x, y)$, using each of the estimated rigid motions. We obtain the 3D coordinate of each pixel in the previous

image, or $\boldsymbol{P}_i$, by using Eq. (2.2) and $D_0(x, y)$. Given the j^{th} estimated rigid motion, we first compute $\boldsymbol{P}'_i$ by applying the rigid motion, from which the reprojected image, denoted as $I^j(x, y)$, can be defined as follows:

$$I^j(x'_i, y'_i) = I_0(x_i, y_i) \tag{4.9}$$

where $(x'_i, y'_i)^T = \pi(\boldsymbol{P}'_i)$. Here, we use the superscript of I to represent the index of the pose used to reproject the previous image, whereas the subscript of I is used to indicate the temporal relation between images. The photometric error is the absolute difference between this reprojected image and the current one, $I_1(x, y)$, and we define it as:

$$E^j(x_i, y_i) = \left| I_1(x'_i, y'_i) - I^j(x'_i, y'_i) \right| \tag{4.10}$$

with (x'_i, y'_i) similarly defined. To ensure that the photometric error is locally smooth, we also filter $E^j(x, y)$ with a guided filter [29] and use $I_0(x, y)$ as the guide image. We show examples of the reprojected image and the resulting photometric error in Figure 4-8.

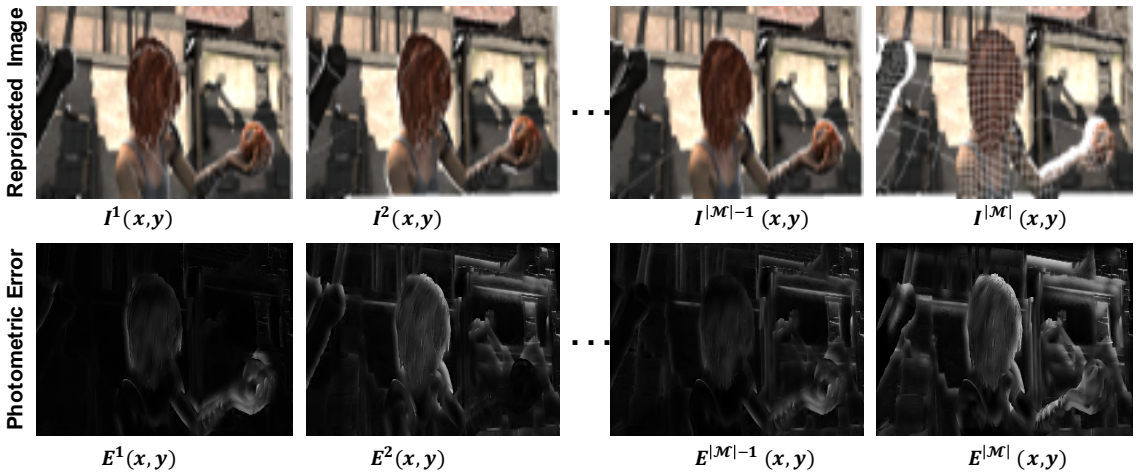


Figure 4-8: **Photometric Error**: We reproject the previous image using the previous depth map and the estimated rigid motions as shown in the first row. The photometric error, shown in the second row, is then obtained by computing the absolute difference between the reprojected images and the current one and filtering this output using a guided filter.

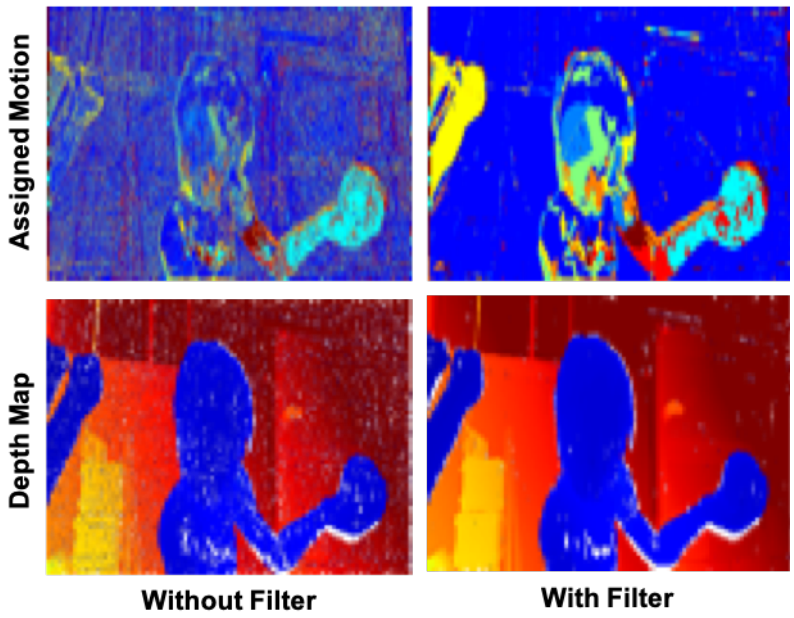


Figure 4-9: **Motion Assignment**: We show the motion assignment, where different colors represent the different estimated rigid motions, and the resulting depth map. Without filtering, the estimated rigid motions are spuriously assigned and results in artifacts in the depth map. When the photometric error is filtered, we see that the motion assignment is locally smooth and the depth map contains fewer artifacts.

Finally, we assign the j^{th} estimated rigid motion to the i^{th} pixel if it minimizes the photometric error. We express this mathematically as follows:

$$j = \underset{1 \leq k \leq |\mathcal{M}|}{\operatorname{argmin}} \; E^k(x_i, y_i) \tag{4.11}$$

We solve Eq. (4.11) for every pixel in the previous depth map, and we visualize an example of this motion assignment in Figure 4-9. This figure also shows the impact of the guided filtering. In our experiments, we find that filtering the photometric error is essential and helps ensure that the motion assignment is also locally smooth, which agrees with our intuition that dynamic scenes are locally rigid. Furthermore, we also see that the resulting depth maps also have fewer artifacts, and we discuss the cause of these artifacts in Section 4.6.5. Our approach is similar to the framework proposed in [74], which applied the same filtering operations for optical flow estimation and stereo matching.

In the process of assigning the estimated rigid motions, our algorithm also seg-

ments the rigid motion in the scene. This is in contrast to some of the approaches in Section 4.2, which require dense optical flow estimation and rigid motion segmentation before depth can be estimated. We are able to do this without these operations because we have a previous depth map, which allows us to compute the photometric error to determine the best rigid motion assignment. Furthermore, this process is also computationally efficient. Computing and filtering the photometric error in Eq. (4.10) has $O(n)$ complexity, where n is the total number of pixels in the previous depth map. This is repeated for each of the $|\mathcal{M}|$ estimated rigid motions, and this computation can be parallelized. We note that $|\mathcal{M}| \leq \frac{N}{N_{\min}}$, where N and $N_{\min}$ are the parameters used to estimate the rigid motions in Section 4.5.1. In our experiments, we choose these parameters to balance the accuracy of the estimated depth maps with its throughput. As a result of this, we are able to estimate dense depth maps in real-time, or under 33.3 milliseconds for each depth map, on a standard laptop computer (2.7 GHz i5-5257U cores) compared to the *minutes* reported by the methods summarized in Section 4.2.

Reprojection

Finally, once the estimated rigid motions are assigned, we estimate the current depth map by reprojecting the previous one. For every pixel in the previous depth map, we first compute its 3D coordinate, $\boldsymbol{P_i}$, using Eq. (2.2) and then compute $\boldsymbol{P_i}'$ using its rigid motion determined by Eq. (4.11). The depth map is finally obtained as follows:

$$D_1(x', y') = \hat{\boldsymbol{z}} \cdot \boldsymbol{P_i}' \tag{4.12}$$

where $(x', y')^T = \pi(\boldsymbol{P_i}')$. When multiple 3D points are reprojected to the same pixel location, we retain the smaller depth value. We summarize our depth estimation process in Algorithm 4.

Algorithm 4 Depth Map Estimation

input: $I_0(x, y), I_1(x, y)$, and $D_0(x, y)$; $\mathcal{M} = \{(\omega_i, t_i)\}$
output: $D_1(x, y)$
$j \leftarrow 1$
repeat
    Reproject $I_0(x, y)$ using $\boldsymbol{\omega_j}, \boldsymbol{T_j}$, and $D_0(x, y)$
    Compute and filter $E_j(x, y)$ using Eq. (4.10)
    $j \leftarrow j + 1$
until $j = |\mathcal{M}|$
repeat
    Compute the best motion j using Eq. (4.11)
    Compute $\boldsymbol{P_i'}$ using $\boldsymbol{P_i}$, $\boldsymbol{\omega_j}$ and $\boldsymbol{T_j}$ with Eq. (4.7)
    $(x_i', y_i')^T \leftarrow \pi(\boldsymbol{P_i'})$
    $D_1(x_i', y_i') \leftarrow \hat{\boldsymbol{z}} \cdot \boldsymbol{P_i'}$
until all pixels are reprojected

4.6 Part II: Evaluation for Dynamic Scenes

4.6.1 Methodology

To evaluate our algorithm, we use RGB-D datasets that contain calibrated image and dense depth map pairs of different dynamic scenes. These datasets are also used to evaluate the approaches in Section 4.2 and include:

- **Deformable Surfaces (DS)** [87]: This dataset contains real sequences of objects undergoing non-rigid deformations. We use the *kinect_paper* and *kinect_tshirt* sequences. We also used this dataset previously to evaluate our algorithm for non-rigid objects.

- **MPI Sintel (MPI)** [7]: This dataset contains synthetic scenes with both articulated and camera motion. We use the clean video sequences of *alley_1*, *ambush_7*, *bandage_1*, *bandage_2*, *shaman_2*, *shaman_3*, *sleeping_1*, and *sleeping_2*.

- **TU Munich RGB-D (TUM)** [85]: This dataset is typically used to benchmark SLAM algorithms. We use the sequences in the *Dynamic Objects* category, which contain both camera and human motion.

- **Virtual KITTI (VKITTI)** [21]: This dataset contains synthetic scenes from the perspective of a car driving through different urban environments. We use the overcast sequences for *1*, *2*, *6*, *18*, and *20*.

We test our approach by estimating depth maps sequentially: we first estimate a new depth map using its corresponding image and a previous image and measured depth map pair, and for consecutive frames, we then use the *estimated* depth map to obtain the next one.

To evaluate the estimated depth maps, we compute the percent mean relative error (MRE) as defined in Eq. (1.10). This metric penalizes a unit error at a close range more than that further away, which is an appropriate metric for applications that use depth to interact with its immediate environment. The MRE also allows us to compare the performance of our algorithm across datasets that have different dynamic ranges. To highlight the different dynamic ranges for the different datasets, we also compute the mean absolute error (MAE) and the root mean squared error (RMSE). These metrics are defined as follows:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|Z_i - \hat{Z}_i| \tag{4.13}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Z_i - \hat{Z}_i)^2} \tag{4.14}$$

where $\hat{Z}_i$ and Z_i are the estimated and measured depth for the i^{th} pixel, respectively, and N is the total number of depth estimates. In our evaluation, we compute these metrics for pixels with ground truth depth values that are within 20 meters. We compute these metrics for 10 sequentially estimated depth maps and average them over 100 different starting points for the sequences in each dataset (except the MPI dataset, where each sequence only has 50 frames).

4.6.2 Implementation

We implement our algorithm following the details stated in Sections 4.5.1 and 4.5.2 and tune algorithm parameters separately for each dataset. Whenever possible, we use OpenCV to implement our approach. As shown in Table 4.4, our code estimates dense depth maps in real-time, or over 30 frames per second (FPS), for the DS and TUM datasets and in near real-time for the other datasets on a standard laptop computer (2.7 GHz i5-5257U cores). This is significantly faster than the approaches in Section 4.2, which report *several minutes* to estimate depth maps of the same resolution. Our computation is dominated by the motion assignment, where approximately 75% of the time is spent on assigning the estimated rigid motions, and the remaining 25% of the time is spent on estimating the rigid motions.

Dataset	**Resolution**	**Frame Rate (FPS)**	**Time Per Frame (ms)**
DS	640×480	32	31.3
MPI	1024×436	12	83.3
TUM	640×480	34	29.4
VKITTI	1242×375	14	71.4

Table 4.4: **Throughput**: We summarize the median frame rate and the estimation time per frame for the sequences of each dataset as profiled on a standard laptop computer (2.7 GHz i5-5257U cores). This is significantly faster than previous approaches, like Kumar *et al.* [46], which require *several minutes* to estimate a 1024×436 depth map.

4.6.3 Results

The results of our evaluation are shown in Figures 4-10 to 4-12, where we plot the MRE, MAE, and RMSE for the depth maps as they are sequentially estimated. Additionally, we also average the MRE, MAE, and RMSE over the frames for each dataset in Table 4.5 and show examples of the estimated depth maps in Figure 4-13.

We observe that all of the error metrics increase as more consecutive depth maps are estimated, and this is due to the errors in the motion estimation and assignment that accumulate across frames. This is especially pronounced in the TUM and

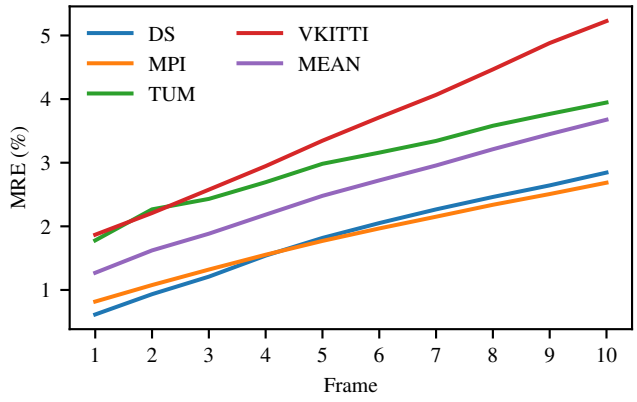


Figure 4-10: **Sequential Estimation Results**: The average MRE of consecutively estimated depth maps are plotted for the datasets we evaluate on.

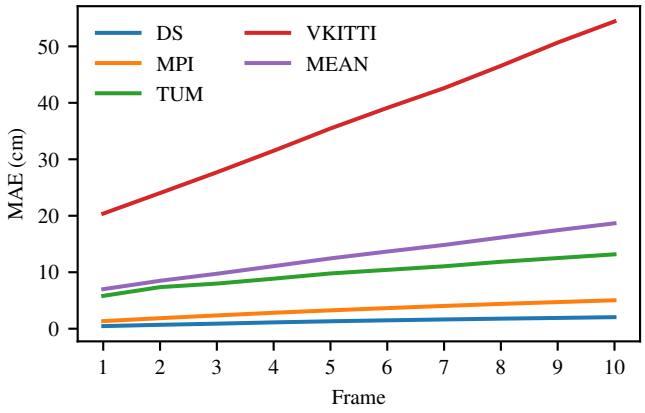


Figure 4-11: **Sequential Estimation Results**: The average MAE of consecutively estimated depth maps are plotted for the datasets we evaluate on.

VKITTI datasets, which have higher errors than the DS and MPI datasets. For the TUM dataset, the images are affected by motion blur and are not perfectly time synchronized with the depth maps. This impacts the motion assignment as described in Section 4.5.2. For the VKITTI dataset, the difference between the foreground and background depth is large. Therefore, errors at these boundaries between the foreground and background (due to erroneous motion assignments, for example) are high.

Nonetheless, our algorithm is still accurate, and when averaged across the different datasets, we see that our algorithm obtains a MRE between 1.3%-3.7%. Moreover, our results also suggest that we can reduce the usage of the depth sensor by over 90%

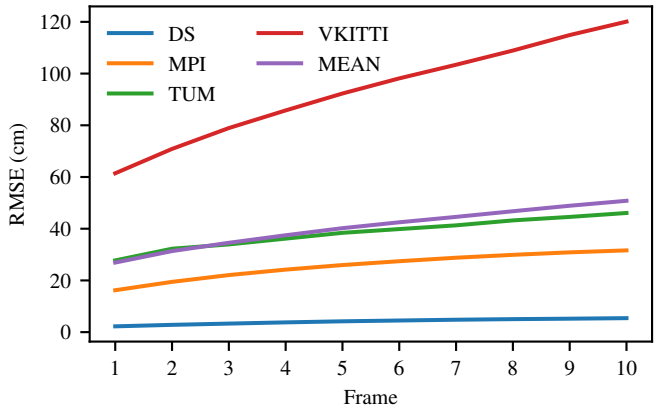


Figure 4-12: **Sequential Estimation Results**: The average RMSE of consecutively estimated depth maps are plotted for the datasets we evaluate on.

Dataset	MRE (%)	MAE (cm)	RMSE (cm)
DS	1.8	1.3	4.1
MPI	1.8	3.3	25.6
TUM	3.0	9.9	38.4
VKITTI	3.5	37.2	93.5
Mean	2.5	12.9	40.4

Table 4.5: **Results for Dynamic Scenes**: We summarize the performance of our approach on each dataset by averaging the different metrics over the frames we estimate.

but still estimate depth maps within 2.5% of the ground truth for general scenes.

4.6.4 Comparison to Previous Approaches

To better evaluate our approach, we compare its performance to the approaches in Section 4.2. We use the depth transfer methods, namely Wang *et al.* [89] and Karsch *et al.* [43], because they have the most similar setup. For the non-rigid structure-from-motion techniques, we compare our technique to Kumar *et al.* [46], which estimates dense depth using consecutive frames without segmentation. We do not compare our technique to the neural network-based approaches because these networks are trained on images with different resolutions and characteristics compared to those in the datasets we evaluate on as that would negatively bias its performance.

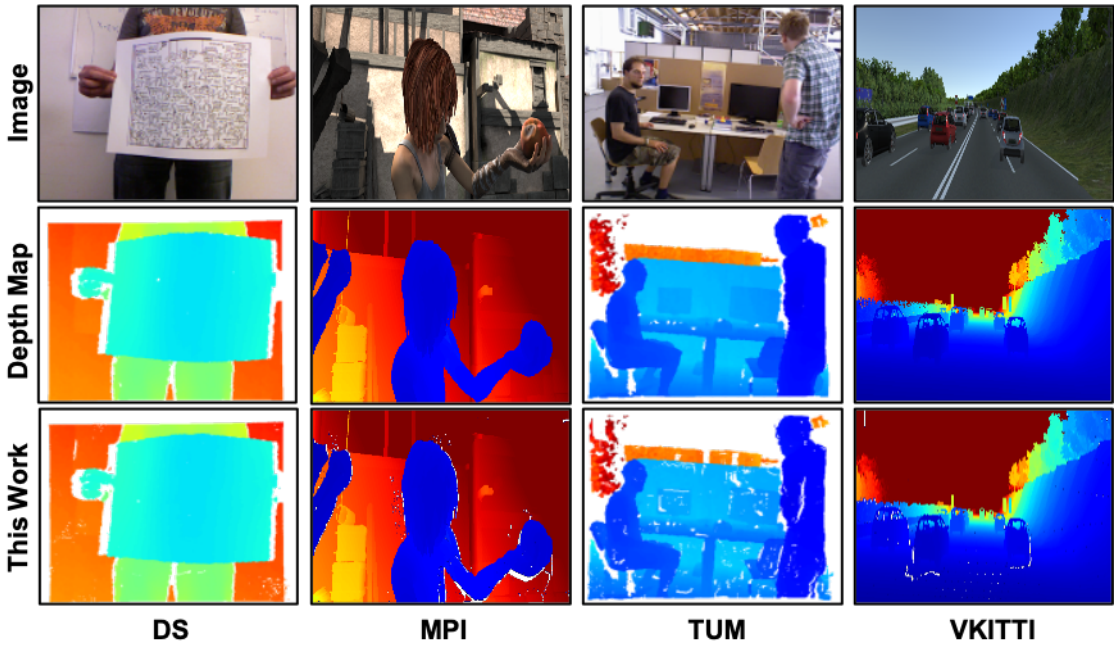


Figure 4-13: **Estimated Depth Maps**: We compare our estimated depth maps (This Work) against the ground truth. The white areas indicate regions where depth is unavailable. We discuss the cause of these regions in our estimated depth maps in Section 4.6.5.

Comparison to Depth Transfer Techniques

We compare our approach to a causal variant of Wang *et al.* [89], which only uses the previously measured depth maps to estimate a new one. As stated in Section 4.2, this method would be effective for small changes in depth and in-plane motion. In Figure 4-14, we see that the MRE for Wang *et al.* [89] increases significantly from frames to frame. This suggests substantial changes in depth in the scenes, which the dense optical flow cannot account for, but is captured by our approach. Consequently, as shown in Table 4.6, our technique outperforms this approach for every dataset we evaluate on.

Karsch *et al.* [43] estimates depth maps by querying from a training set of image and depth map pairs captured from similar scenes. For our experiments, we randomly selected image and depth map pairs from each dataset for this technique to use. We use the default parameters, where each depth map is estimated using 8 examples from the training set. However, even with a training set taken from the same scenes, we

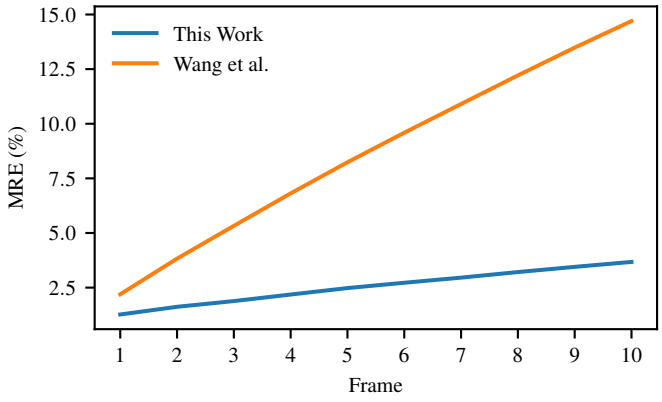


Figure 4-14: **Comparison to Depth Transfer Techniques**: Our technique not only remaps the pixels of a previous depth maps but also accounts for the changes in depth.

Dataset	**This Work**	**Wang *et al.*** [89]	**Karsch *et al.*** [43]	**Kumar *et al.*** [46]
DS	1.8	4.6	6.9	4.8
MPI	1.8	3.6	13.9	16.7
TUM	3.0	6.0	18.5	-
VKITTI	3.5	20.6	14.9	10.45

Table 4.6: **Comparison to Previous Approaches**: We summarize the MRE of the depth maps estimated by our algorithm and the previous techniques. The code for Kumar *et al.* [46] is not publicly available, and we report the results from their paper.

see in Table 4.6 that our approach outperforms Karsch *et al.* [43]. This makes sense because similar images of the same scenes are not guaranteed to have the same depth, and this is reflected by the high MRE.

Comparison to Non-Rigid Structure-from-Motion

We also compare our approach to Kumar *et al.* [46], which is similar to our technique in that it estimates depth maps using only two frames and assumes that the scene is locally rigid. This approach is different in that it does not have any previous depth. Therefore in order to estimate depth, it first oversegments the scene into rigid superpixels and then uses the *per-pixel*, dense optical flow to estimate the depth within each superpixel while ensuring scale consistency. This requires restrictive assumptions for each superpixel and is sensitive to the accuracy of the optical flow, which is not

always possible to accurately estimate. While this method produces encouraging results (especially since it does not use any previous depth measurements like in our approach and the previous ones), our technique still outperforms it as shown in Table 4.6. This shows that using prior depth allows us to both efficiently and accurately estimate depth in dynamic scenes.

4.6.5 Limitations of Our Approach

As described in Section 4.5.1, our algorithm estimates the independent and rigid motions in the scene using the optical flow at a sparse set of key points. Therefore, our approach will naturally fail for scenes with limited texture. Another consequence of this is that our approach also fails to estimate the motion of small rigid segments, where key points are not detected and the optical flow not estimated. One way to detect these scenarios is to examine the photometric error and use it as a measure of confidence for the estimated depth, an area of future exploration.

Another drawback of our approach stems from the motion assignment and depth map estimation in Section 4.5.2. For regions with limited texture, the pose can be incorrectly assigned and lead to pixels being erroneously reprojected. However, even when the pose is correctly assigned, we still have missing depth estimates in the depth map. Some of these missing pixels arise because reprojecting the neighboring pixels of a rigid segment does not constrain them to be contiguous in the estimated depth map. However, these missing pixels are smaller in area compared to those in regions which were previously occluded, but uncovered due to the motion in the scene. This is shown in Figure 4-15. Furthermore, as the depth maps are sequentially estimated, these holes become more pronounced. While there are many promising infilling approaches [8, 32, 53, 58, 72], we avoid them because they often require assumptions that are not geometrically motivated. We can also avoid these infilling techniques because in our problem setup, we can still use the depth sensor. For example, when the depth in a previously occluded region is required by the underlying application, this event can be used to trigger the depth sensor to obtain a new depth map as done in [64].

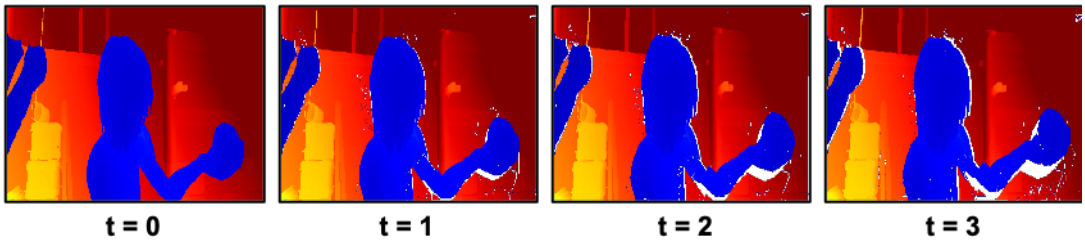


Figure 4-15: **Missing Depth**: As the hand moves, part of the background becomes unoccluded, and because its depth was not previously measured, they are missing in the reprojected depth map.

4.7 Summary

In this chapter, we present algorithms that reduce the usage of ToF cameras and estimate new depth maps using concurrently collected RGB images for non-rigid objects and dynamic scenes. We present solutions that balance the acquisition of depth between the ToF camera and computation without incurring a large cost. Our contribution is to incorporate the previous depth measurements into our formulation, which allows us to efficiently estimate the rigid motions in the scene using sparse optical flow and to accurately assign the estimated rigid motions to the pixels of the previous depth map to obtain a new one. The resulting algorithms are efficient and, in the case of dynamic scenes, we estimate dense depth maps in up to real-time (30 FPS) on a standard laptop computer. Across different datasets, we show that our techniques can reduce the usage of the ToF camera by up to 90% but still estimate depth maps with an average mean relative error of 2.5%, outperforming related techniques.

Chapter 5

Low Power Temporal Depth Filtering

5.1 Introduction

In the previous chapters, we assumed that RGB images are concurrently collected, and we used them to reduce the usage and sensor power of the time-of-flight (ToF) camera. To maintain the acquisition of depth, we then estimated new depth maps using the RGB images. Here, we examine how we can reduce the sensor power using *only the data from the ToF camera*. To do so, we take a different approach and focus instead on obtaining accurate depth maps while reducing the overall amount of emitted light. In contrast to the previous settings, the ToF camera is always on in this scenario.

Naturally, one simple way to reduce the sensor power is to reduce the amount of light the ToF camera emits per frame. As a result, the intensity of the reflected light that is captured by the ToF camera is also reduced. One consequence of this is the loss of range because the reflection from distant objects cannot be discerned from the ambient light. Even for applications that only need depth in the immediate vicinity, this simple strategy is still unappealing because the reduced intensity also increases the noise in the resulting depth map.

To address this limitation, we propose algorithms that reduce the sensor power of a ToF camera and mitigate the noise in the low power depth maps by combining the data across consecutive frames. To do so, our algorithms use the infrared (IR) images

that a ToF camera obtains in place of the RGB images to estimate the 3D scene motion using the techniques described in the preceding chapters. This 3D motion can be estimated efficiently using the optical flow and depth at a *sparse set of pixels*. In particular, we propose the following:

- **Adaptive Power Depth Map Denoising**: To reduce the overall sensor power of the ToF camera, we reduce the frequency in which high power depth maps are obtained and use them to denoise subsequent low power depth maps for *dynamic scenes*. Our key insight here is that the 3D scene motion can be estimated using infrared images obtained under different power settings, and we use the estimated rigid motions to combine the high power depth maps with the low power ones sequentially. The resulting algorithm reduces the mean relative error of the low power depth maps by up to 64% and the overall amount of emitted light by up to 81%.

- **Recursive Temporal Filtering**: When only low power depth maps are collected, we propose an algorithm to temporally filter these depth maps recursively. Because the range of a low power depth maps is reduced, we assume that the region that the ToF camera can sense is *rigid*. Our technique estimates this rigid motion and uses it to efficiently align consecutive depth maps spatially and account for the changes in depth. The resulting filter reduces the mean relative error of the low power depth maps by 77%.

By exploiting sparse and recursive processing, both algorithms filter depth maps in up to real-time using the CPU of an embedded processor (Cortex-A7/A15 octa core).

This chapter is based on [61] and is organized as follows. In Section 5.2, we analyze the noise in low power depth maps that our methods aim to mitigate. The remainder of the chapter is divided between our two approaches. In Part I (Sections 5.3 to 5.5), we describe our adaptive power depth map denoising algorithm. In Section 5.3, we provide an overview of the assumptions we made for this approach. We then describe the algorithm in Section 5.4 and evaluate it in Section 5.5. In Part II (Sections 5.6 to 5.8), we describe our recursive temporal filtering approach. In Section 5.6, we

motivate the need for this approach and describe the assumptions we made. We then describe the filtering approach in Section 5.7 and evaluate it in Section 5.8. Finally, we conclude this chapter in Section 5.9.

5.2 Shot Noise in Time-of-Flight Cameras

As described in Chapter 1, ToF cameras estimate depth by emitting pulses of light and accumulating its reflection over carefully-timed intervals. Due to the discrete nature of light, the amount of light that is reflected and accumulated by the sensor is affected by shot noise, which is also known as Poisson noise [40]. To see how this affects the resulting depth map, we analyze its impact on a pulsed ToF camera.

Pulsed ToF cameras obtain depth by estimating the round trip time of the emitted pulses of light. We show the operation of a pulsed ToF camera in Figure 5-1 for a single pulse and pixel. In the figure, we indicate the interval in which the light is emitted as well as the intervals, denoted as *Shutter 0* and *Shutter 1*, where the reflected light is accumulated. To simplify our analysis, we will assume that there is no ambient contribution or multi-path interference.

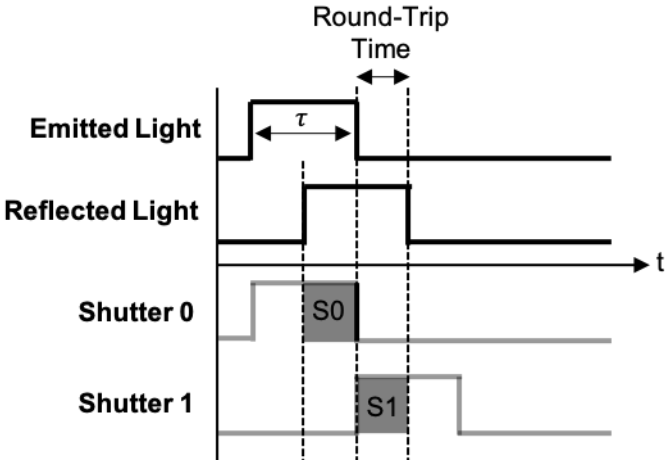


Figure 5-1: **Pulsed Time-of-Flight Camera:** We depict the operation of a pulsed ToF camera for a single pulse and pixel. In particular, we show the time interval where the pulse of light is emitted and the intervals (e.g., Shutter 0 and Shutter 1) where the reflected light pulse is accumulated.

To estimate the depth for the i^{th} pixel, we observe that the light accumulated

in *Shutter 1*, which we denote as $S1_i$, is proportional to the round trip time of the emitted light. Furthermore, the light accumulated in both *Shutter 0* and *Shutter 1*, or $S0_i + S1_i$, is proportional to the entire pulse width. This allows us to obtain the depth, denoted as Z_i, as follows:

$$Z_i = \frac{c\tau f}{2\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2 + f^2}} \frac{S1_i}{S0_i + S1_i} \tag{5.1}$$

where f and (x_c, y_c) are the intrinsic parameters, τ is the pulse width of the emitted light, and c is the speed of light. This is repeated for every pixel on the image sensor to obtain a depth map, or $D(x, y)$. These same measurements can also be used to obtain an IR image, or $I(x, y)$, where the intensity of each pixel is proportional to the light accumulated in Shutter 0 and Shutter 1.

Because the accumulated light is affected by shot noise, the process shown in Figure 5-1 is often repeated for many pulses to obtain reliable depth estimates [25]. However, when the amount of the accumulated light is still low, which is the case when the amount of emitted light is reduced, the impact of shot noise is pronounced in the resulting depth map. To see this, we approximate the variance of the estimated depth, denoted as $\sigma^2_{Z_i}$, by using the propagation of uncertainty method:

$$\sigma^2_{Z_i} \approx \left(\frac{\partial Z_i}{\partial S0_i}\right)^2 \sigma^2_{S0_i} + \left(\frac{\partial Z_i}{\partial S1_i}\right)^2 \sigma^2_{S1_i} \tag{5.2}$$

where $\sigma^2_{S0_i}$ and $\sigma^2_{S1_i}$ are the variances of $S0_i$ and $S1_i$, respectively.

Assuming that shot noise is the dominant noise source, we model $S0_i$ and $S1_i$ as scaled and independent Poisson random variables (e.g., $S0_i/\alpha$ and $S1_i/\alpha$ are Poisson distributed for some constant α) to obtain $\sigma^2_{S0_i}$ and $\sigma^2_{S1_i}$. Using this fact and by substituting Eq. (5.1) into Eq. (5.2), we can rearrange the terms to obtain the following expression:

$$\sigma^2_{Z_i} \approx Z_i \left(\frac{c\tau f}{2\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2 + f^2}} - Z_i\right) \frac{\alpha}{S0_i + S1_i} \tag{5.3}$$

where we see that $\sigma^2_{Z_i}$ is inversely proportional to the intensity of the reflected light, or $S0_i + S1_i$. Therefore, when this intensity is low, the depth variance is especially high, and a consequence of this is that the depth resolution is reduced. A similar expression can also be derived for continuous wave ToF cameras following the same procedure.

The high noise in low power depth maps shows why the simple approach of just reducing the amount of emitted light is insufficient for our goal. This motivates the approaches that we propose to increase the accuracy of the low power depth maps. It should also be noted that in addition to shot noise, the depth maps obtained by a ToF camera are also affected by other statistical and systematic noise sources. In this chapter, we only focus on shot noise as it is especially pronounced in low power depth maps.

5.3 Part I: Adaptive Power Depth Maps Denoising

In this section, we present an algorithm that obtains adaptive power depth maps and lowers the overall power required to obtain accurate depth maps for a ToF camera. Our algorithm accomplishes this by reducing the frequency in which high power depth maps are obtained and using the high power depth maps to denoise subsequent low power ones. This is shown in Figure 5-2.

In order to denoise the low power depth maps, our algorithms estimate the 3D motion between the consecutive frames. We show that this 3D motion can be estimated using the consecutive IR images, *regardless of whether it was obtained in a high or low power setting.* This is because features are still preserved in the IR images, which is essential for the estimation of optical flow. In our algorithm, we use these IR images to estimate the motion in *dynamic scenes* by adapting the technique in Chapter 4. We describe this adaptation next.

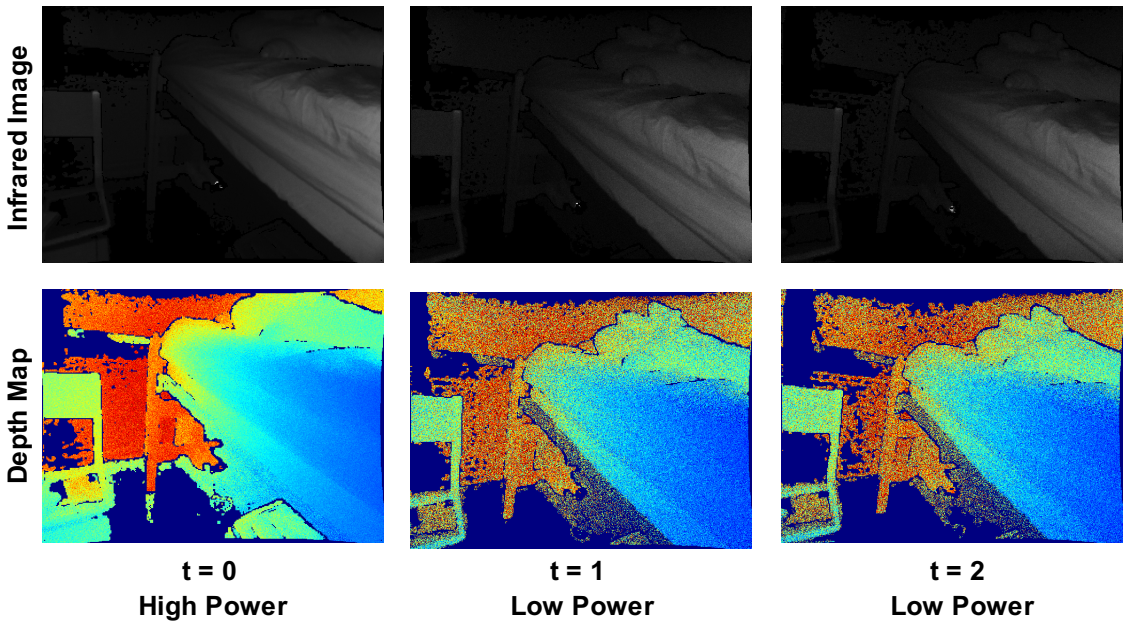


Figure 5-2: **Adaptive Power Depth Map Denoising**: We adaptively vary the power of a ToF camera in each frame. Here, we assume that high power depth maps are obtained with 10× as much power as the low power ones. To increase the accuracy of the first low power depth map ($t = 1$), we estimate and use the 3D scene motion to combine the previous high power depth map with it. We repeat this sequentially to increase the accuracy of the next low power depth map ($t = 2$).

5.4 Part I: Proposed Algorithm

To increase the accuracy of the low power depth maps, our algorithm takes as input the previous IR image and depth map along with the current IR image and depth map. We denote the previous IR image and depth map as $I_{t-1}(x, y)$ and $\hat{D}_{t-1}(x, y)$, respectively. We use the $\hat{D}(x, y)$ notation to indicate that the depth map has either been filtered or obtained in a high power setting. We denote the current IR image and depth map as $I_t(x, y)$ and $D_t(x, y)$, respectively. We assume that $D_t(x, y)$ is a low power depth map, and our goal is to filter it to obtain $\hat{D}_t(x, y)$.

As shown in Figure 5-3, our algorithm accomplishes this by first estimating the 3D motion between the consecutive frames. When the motion can be estimated, our algorithm combines the previous depth map with the current one. However, when the motion cannot be estimated, we obtain a new high power depth map using the ToF camera.

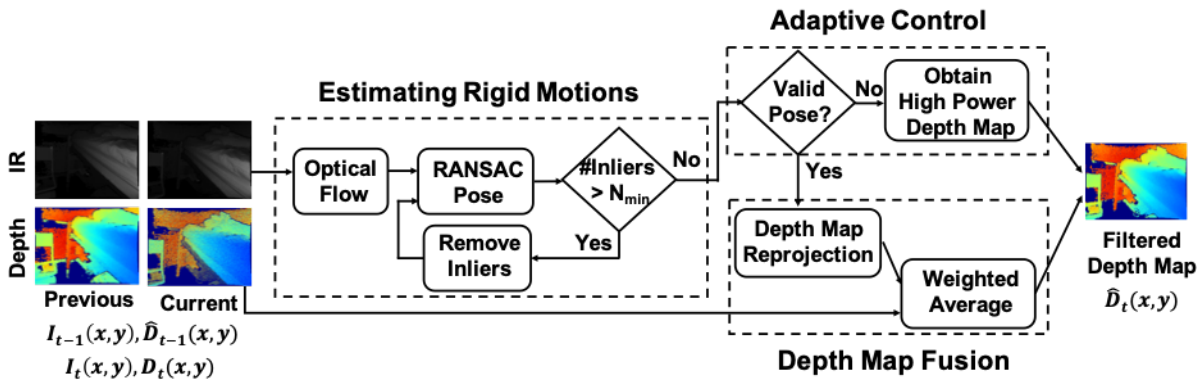


Figure 5-3: **Adaptive Power Depth Map Denoising Pipeline**: Our algorithm increases the accuracy of the current low power depth map by combining it with a previously filtered or high power one. It does this efficiently by estimating the 3D motion in the scene, and when this motion cannot be estimated, it signals the ToF camera to obtain another high power depth map.

5.4.1 Estimating Rigid Motions

Similar to Chapter 4, we model the dynamic scene using independent rigid motions. To estimate these motions, we follow an approach similar to that described in Section 4.5.1 by clustering the rigid motions in order of their inlier set sizes using the sparse optical flow computed across the consecutive IR images, $I_{t-1}(x, y)$ and $I_t(x, y)$, and the corresponding depth taken from the previous depth map, $\hat{D}_{t-1}(x, y)$.

Here, we represent the rigid motion using the pose, which is composed of a rotation matrix and a 3D vector. We also compute the optical flow for the pixels on a sparse and uniformly spaced grid. This minimizes the computation because we do not need to localize key points. For the scenes that we consider, which contain a small number of independently moving objects, this is a reasonable tradeoff. We also estimate optical flow using block matching with normalized correlation. In our experiments, we found that performing block matching using 15×15 blocks centered on the pixels of a 20×20 sparse grid offered the best tradeoff between the accuracy of the filtered depth map and the latency in which they are obtained. With the poses estimated, we use them to combine the previous depth map with the current one.

5.4.2 Depth Map Fusion

To denoise the current depth map, we first need to assign the rigid motions to the appropriate pixels of the previous depth map. Once the rigid motions are assigned, we then reproject the previous depth map and combine it with the current depth map using a weighted average.

Depth Map Reprojection

To assign the rigid motions to the pixel of the previous depth map, the approach in Section 4.5.2 reprojects the previous image and uses the photometric error (e.g., the absolute difference between the reprojected image and the current one) to guide the rigid motion assignment. Here, we take a different approach and instead reproject the *previous depth map* and compare it to current depth map. By using the current depth map, we will show that we can avoid the overhead of reprojecting the entire depth map for each of the estimated poses.

As shown in Figure 5-4, we first reproject the previous depth map using the pose with the largest inlier set. We assume that this pose corresponds to the largest rigid region in the image. We first obtain the 3D position of each pixel in the previous depth map, which we denote as $\boldsymbol{P_i}$, by using Eq. (2.2) and $\hat{D}_{t-1}(x, y)$. We then apply the pose, which we denote as $\boldsymbol{R_0}$ and $\boldsymbol{T_0}$, to each point and obtain its new pixel coordinate, (x'_i, y'_i), and updated depth Z'_i, as follows:

$$(x'_i, y'_i)^T = \pi\left(\boldsymbol{R_0 P_i} + \boldsymbol{T_0}\right) \qquad Z'_i = \hat{\boldsymbol{z}} \cdot \left(\boldsymbol{R_0 P_i} + \boldsymbol{T_0}\right) \tag{5.4}$$

where $\pi(\cdot)$ is the projection operator defined in Section 2.3.

As shown in Figure 5-4, we only retain the updated depth where $|Z'_i - D_t(x'_i, y'_i)| < \epsilon$, where ϵ is a depth threshold. For these pixels, we update the reprojected depth map, denoted as $\tilde{D}_{t-1}(x, y)$, as follows:

$$\tilde{D}_{t-1}(x'_i, x'_i) = Z'_i \tag{5.5}$$

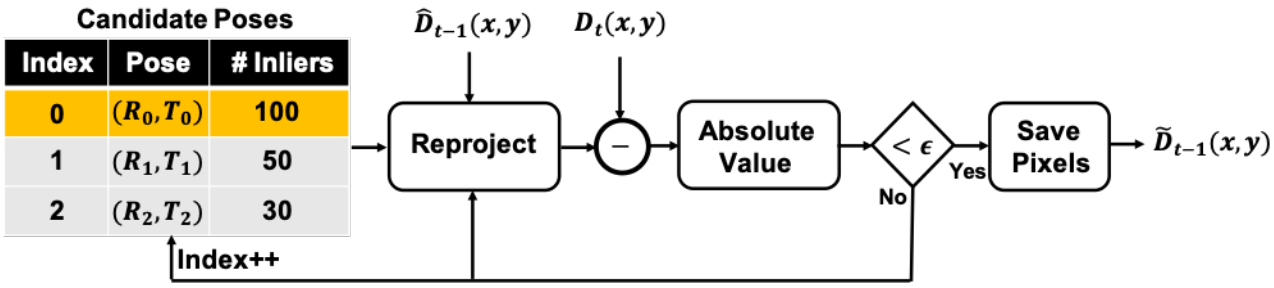


Figure 5-4: **Depth Map Reprojection Pipeline**: We apply the poses in order of its inlier set size and use the current depth map to help assign the correct pose to each pixel.

We repeat this process with the remaining pixels (e.g., where $|Z_i' - D_t(x_i', y_i')| \geq \epsilon$) using the remaining poses in order of their inlier set sizes. Therefore, this approach reprojects *fewer pixels* than the approach in Section 4.5.2, which reprojects *every pixel* in the previous image for each pose. Even though the current low power depth map is inaccurate, we find that it still can be used to guide the pose assignment.

We show examples of the pose assignment for 2 scenarios in Figure 5-5. In the third column of this figure, we color code each pixel according to its assigned pose. We see that the pose assignment is consistent with the segments that have different motions. This qualitatively shows that our approach estimates and correctly assigns the motion.

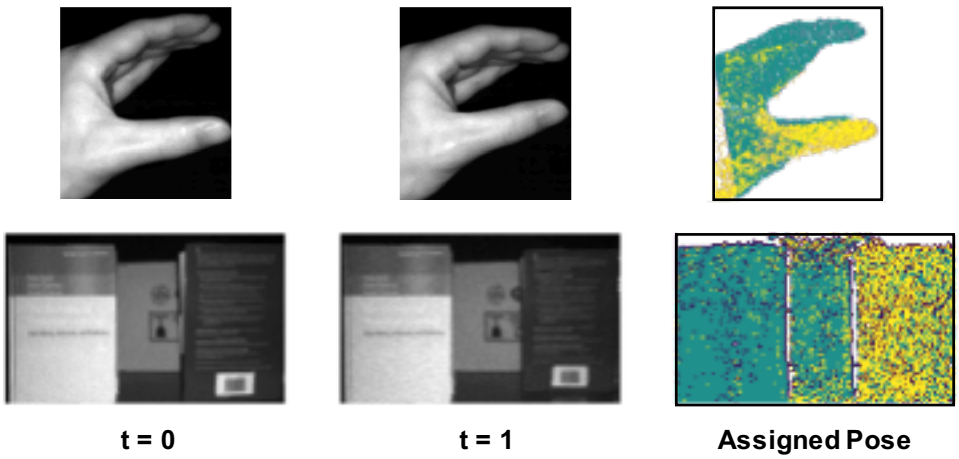


Figure 5-5: **Assigned Pose**: We show examples of the assigned pose, which is consistent with the rigid object motion in the scene.

Weighted Average

To obtain $\hat{D}_t(x, y)$, we combine the non-zero pixels of the reprojected depth map with the current one using a weighted average. It can be shown that choosing the weights based on the variances of the depth lowers the expected mean squared error. Therefore, we obtain the filtered depth map as follows:

$$\hat{D}_t(x, y) = \frac{\sigma_t^2 \tilde{D}_{t-1}(x, y) + \sigma_{t-1}^2 D_t(x, y)}{\sigma_t^2 + \sigma_{t-1}^2} \tag{5.6}$$

where we denote σ_{t-1}^2 as the variance of $\tilde{D}_{t-1}(x, y)$ and σ_t^2 as the variance of $D_t(x, y)$. These variances must be computed for *every pixel* using Eq. (5.2) and also updated as the pixels are reprojected. To reduce this computation, we approximate the weights as follows:

$$\frac{\sigma_t^2}{\sigma_t^2 + \sigma_{t-1}^2} \approx \frac{M}{M + 1} \tag{5.7}$$

where M is the ratio of the power used to obtain a high power depth map to that of a low power one. This approximation shows that more weight is given to the reprojected depth, which makes sense because it contains information from the last high power depth map. Furthermore, in our experiments, we find that using the approximation in Eq. (5.7) had a negligible impact on the mean relative error of the filtered depth maps when compared to using the actual variances.

5.4.3 Adaptive Control

As shown in Figure 5-3, we obtain a high power depth map when the motion cannot be estimated. This occurs when there are not enough features in the IR images or if accurate optical flow cannot be estimated. We also track the number of pixels that are filtered using Eq. (5.6). In our implementation, if that number falls below 50% of the pixels, we also obtain another high power depth map.

5.5 Part I: Evaluation

5.5.1 Methodology

To evaluate our algorithm, we capture 640×480 IR image and depth map pairs using the Pico Zense ToF camera [68]. In our experiments, the high power depth maps are obtained with $10\times$ as much power as the low power ones. We obtain two datasets as shown in Figure 5-6:

1. *Indoor Sequences*: This dataset is obtained by moving the ToF camera around different indoor environments. We use the captured data as ground truth and simulate high and low power depth maps by adding shot noise to the measurements.

2. *Rail Sequence*: This stop sequence dataset is collected using a rail that moves a calibration target away from the ToF camera. We move the target in 5 cm increments and capture 300 depth maps. We use a single depth map to represent a low power depth map, average 10 for a high power one, and average all 300 for the ground truth.

The rail sequence also tests the performance of our algorithm for scenes with multiple rigid objects since the calibration target moves while its surroundings do not.

Figure 5-6: **Datasets**: We show select frames from the indoor and rail sequences used to evaluate our approach.

We apply our algorithm to these sequences and use the high power depth maps when it is required. For each depth map, we compute the percent mean relative error

(MRE) over the N pixels of the filtered depth map as defined in Eq. (1.10). We also note the percentage of high power depth maps that we use in our approach.

5.5.2 Implementation

We implement our algorithm on the ODROID XU-3 board [26], which is an embedded platform with an Exynos 5422 processor. This processor is used in the Samsung Galaxy S5 [78], and we evaluate our approach using this processor because it is representative of the computational resources available on mobile devices, which our algorithm is designed for. The resulting implementation outputs depth maps in real-time (30 FPS) for the data we evaluate on, and the processing time is spent equally between estimating the rigid motions and the depth map fusion.

5.5.3 Results

We summarize the performance of our algorithm in Table 5.1, where we average the MRE across all of the depth maps. For each dataset, we choose the algorithm parameters, namely ϵ, that minimize the MRE. We see that the MRE of our depth maps are 64% and 63% lower than the MRE of the low power depth maps for the indoor and rail datasets, respectively. Our algorithm also only obtains high power depth maps for 10% and 18% of these frames for the respective sequences. This means that we reduce the power for obtaining the depth maps by up to 81%. An example of our filtered depth map (This Work) is shown in Figure 5-7, where it is compared against the low (Low Power) and high (High Power) power depth maps. We see that our depth map is less noisy and visually sharper than the low power one.

5.5.4 Comparison to Alternative Denoising Strategies

Here, we compare our approach to alternative denoising strategies. This includes spatial filters and obtaining low power depth maps with a constant and equivalent power for each frame.

Configuration	Indoor	Rail
This Work	3.2%	4.3%
High Power	2.6%	3.7%
Low Power	8.8%	11.5%
This Work+Bilateral Filter	2.3%	3.4%
Low Power+Bilateral Filter	6.3%	11.0%
Equivalent Power	6.2%	6.8%

Table 5.1: **Mean Relative Error**: Our adaptive algorithm only obtains high power depth maps for 10% and 18% of the frames for the indoor and rail sequences, respectively.

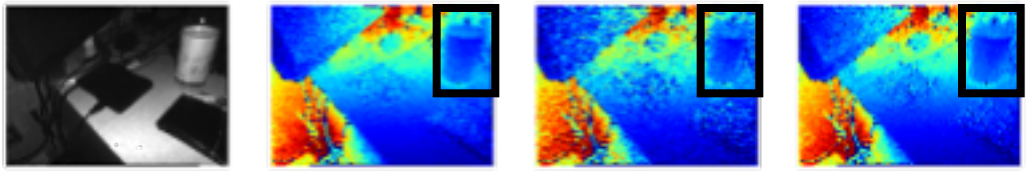


Figure 5-7: **Filtered Depth Map**: We an example of our filtered depth map, which is less noisy than the low power one.

Comparison to Spatial Filters

We compare our depth maps to a low power one that is spatially filtered. A common spatial filter is the bilateral filter [82], and it has been shown to give competitive results [48]. To compare our approach, we apply a 5×5 bilateral filter to the low power depth maps and compute its MRE (Low Power+Bilateral Filter). As shown in Table 5.1, the MRE of the filtered low power depth map remains high. This is because the noise in the low power depth maps causes the depth values to vary significantly from pixel to pixel and appear as edges. Because the bilateral filter is an edge-preserving filter, these pixels remain unprocessed. In contrast, by using the data from a high power depth map, our technique is able to lower the MRE of the low power depth map and outperform the bilateral filter.

Furthermore, it should also be noted that computing the weights of a bilateral filter is not trivial. On the ODROID-XU3 board, applying the bilateral filter lowers the frame rate to 8.6 FPS, which is lower than that of our approach. We can also

apply the bilateral filter to our depth maps (This Work+Bilateral Filter), which further reduces the MRE. However, this would cause the frame rate of our approach to fall below 30 FPS.

Comparison to Equivalent Power

Because our algorithm adaptively alternates between obtaining high power depth maps and low power ones, we also compare our depth maps to those obtained with a constant and equivalent power. We can simulate these depth maps by adding the appropriate noise and averaging the proper number of frames for the indoor and rail sequences, respectively. If these depth maps have a lower MRE, then simply increasing the power for each frame is more desirable than any additional processing. As shown in the Table 5.1, this (Equivalent Power) is not the case, and our approach has a lower MRE. This is because at low power, the impact of noise on the MRE is heavily pronounced.

5.6 Part II: Recursive Temporal Filtering

In this section, we describe an algorithm that lowers the power for depth sensing by obtaining *only low power depth maps* and mitigates the noise in them. In contrast to the previous approach, this algorithm obtains each depth map with the same amount of light. This approach is useful for applications that run on small mobile devices, where the lack of physical space for additional illumination sources limits the amount of light that can be emitted. Furthermore, this does not even have to be a power issue. For example, many applications need depth maps at high frame rates, and as a consequence, the intermediate measurements (e.g., $S0$ and $S1$) used to estimate depth must be obtained at an even higher frame rate. Due to the reduced exposure time, the accumulation of the reflected light is also reduced and increases the noise in the resulting depth maps. Even when the exposure time can be increased, doing so may lead to motion blur when there is motion from frame to frame [25].

Because the low power depth maps have limited range, we will assume that the

regions that the ToF camera can sense is *rigid*. Our algorithm then uses the IR images that a ToF camera obtains to estimate this rigid motion (following the approach in Chapter 3) and uses it to temporally filter the depth maps recursively. Naturally, denoising depth maps is well explored in the literature, and we provide a brief overview of the existing techniques to distinguish our approach from them.

5.6.1 Spatial Filtering

To mitigate noise in depth maps, many different spatial filters are used [10, 20, 38, 49, 67, 82, 93]. These include the standard Gaussian and median filters [20]. To ensure that edges are preserved in the filtered depth maps, both bilateral filters [82] and total variation techniques [49] are also used. Out of these different approaches, the bilateral filter is the most popular [48]. For many applications, images are also typically collected alongside depth maps, and many techniques [10,38,67,93] use them to help filter and enhance the depth maps. In our approach, we will show that the IR images that a ToF camera collects is sufficient to accurately filter depth maps.

While these methods are useful for individual depth maps, in many applications, depth maps are continuously collected, and spatial filters do not exploit this additional information to further reduce noise. Our approach and many other temporal filtering techniques use this redundant data to increase the accuracy of the filtered depth maps. We describe these approaches next.

5.6.2 Temporal Filtering

Temporal filters exploit the fact that neighboring frames are similar and use this redundant information to filter noisy depth maps. When there is no motion between consecutive depth maps, averaging them reduces the depth variance as if *the intensity of the reflected light is increased.* To see this, let $Z_{i,k}$ represent the depth of the i^{th} pixel in the k^{th} depth map. Each $Z_{i,k}$ can be thought of as a sample from the same

distribution. Given this, we see that the variance of the sample mean is equal to:

$$\text{var}\left(\frac{1}{F}\sum_{j=0}^{F-1} Z_{i,k-j}\right) = \frac{\sigma_{Z_i}^2}{F} = Z_i\left(\frac{c\tau f}{2\sqrt{(x_i-x_c)^2+(y_i-y_c)^2+f^2}} - Z_i\right)\frac{\alpha}{F(S0_i+S1_i)} \tag{5.8}$$

where we substitute Eq. (5.2) for $\sigma_{Z_i}^2$ and F is the number of the averaged frames. We see that the denominator in the right-most term of Eq. (5.8) shows that the intensity, $S0_i + S1_i$, is scaled by the number of averaged frames and is equivalent to increasing the intensity of the reflected light.

This concept is leveraged in many temporal filtering approaches including the one proposed here, which we summarize in Table 5.2. Unfortunately, in many applications, there is typically motion between consecutive depth maps. Realizing this, Lin *et al.* [52] average the same pixels across multiple depth maps as long as they are within a threshold of that in the current depth map. To account for the motion between frames, the techniques in [1, 22, 44, 57] compute the optical flow to first align depth maps before they are averaged. In order to compute the optical flow, the techniques in [1,44,57] all use concurrently collected RGB images, which are not available in our problem setup. To reduce the complexity of estimating the dense optical flow, the techniques in [22,57] use block matching. Recognizing that the optical flow does not account for changes in depth along the z-axis, Kim *et al.* [44] uses the optical flow to determine the pixels of stationary objects (e.g. where the optical flow is zero) so that they can be averaged.

Method	**RGB Images**	**Dense Optical Flow**
This Work		
Lin *et al.* [52]		
Georgiev *et al.* [22]		✓
Avetisyan *et al.* [1]	✓	✓
Kim *et al.* [44]	✓	✓
Matyunin *et al.* [57]	✓	✓

Table 5.2: **Comparison of Temporal Filtering Techniques**: We compare the different temporal filtering techniques and use a checkmark to indicate whether the technique needs concurrently collected RGB images or dense optical flow.

In contrast, we take a different approach. Instead of computing the dense optical flow to align the depth maps, we first estimate the pose using just data from the ToF camera. As we have shown in the preceding chapters, the pose can be efficiently estimated using *sparse* correspondences. We then use the pose to directly compute the dense correspondences for the remaining pixels and also account for changes in depth along the z-axis. Furthermore, many of these previous approaches need to store many previous depth maps. Our technique avoids this by instead filtering the depth maps *recursively*. In our evaluation (Section 5.8), we will compare our technique to Lin *et al.* [52] and a variant of the techniques that use optical flow to show the benefits of our design decisions.

5.7 Part II: Proposed Algorithm

In this section, we describe how we temporally filter depth maps *recursively* as shown in Figure 5-8. Our algorithm takes as input the consecutive IR images and depth maps. We denote the previous IR image and depth map as $I_{t-1}(x, y)$ and $\hat{D}_{t-1}(x, y)$, respectively. We use the $\hat{D}(x, y)$ notation to indicate that the depth map has been previously filtered. We denote the current IR image and depth map as $I_t(x, y)$ and $D_t(x, y)$, respectively.

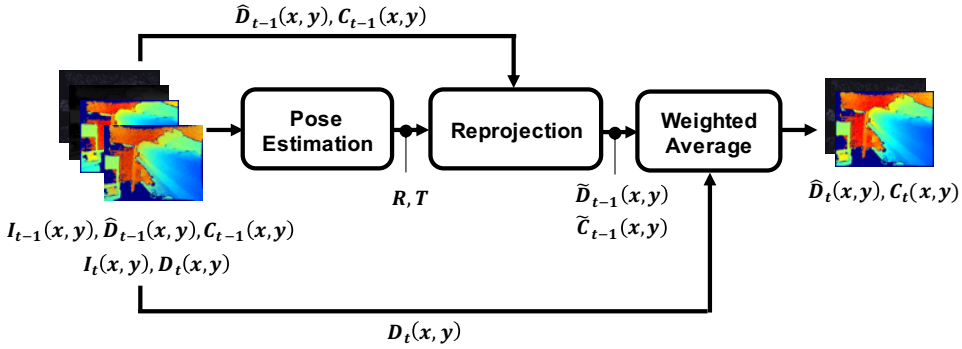


Figure 5-8: **Algorithm Pipeline**: Our algorithm takes as input the current IR image and depth map pair, the previous IR image and filtered depth map pair, a confidence map that stores the number of previous depth maps used to obtain the previously filtered estimates. The output of our approach is the current filtered depth map and an updated confidence map.

In addition, our algorithm also requires a confidence map, $C_{t-1}(x, y)$. For every pixel in the previously filtered depth map, $C_{t-1}(x, y)$ stores the number of depth maps used to obtain the filtered estimate. (Accordingly, we initialize $C_0(x, y) = 1$.) Our algorithm uses $C_{t-1}(x, y)$ to weigh the pixels of the previous depth map. Our algorithm then outputs the filtered depth map, $\hat{D}_t(x, y)$, and an updated confidence map, $C_t(x, y)$. As our approach is recursive, we then use $\hat{D}_t(x, y)$ and $C_t(x, y)$ to filter the next depth map.

5.7.1 Pose Estimation

As shown in Figure 5-8, we first estimate the pose between the consecutive frames using the method described in Section 2.2.3. In our implementation, we preprocess the consecutive IR images by normalizing them so that they have the same mean intensity and dynamic range. As the intensity of an IR image varies as function of depth, this is done so that the brightness constancy assumption is approximately maintained when there is motion along the z-axis, which is important for estimating the optical flow.

Because the IR image and depth map come from the same source, we can estimate the pose accurately without being impacted by errors in their synchronization. Therefore, unlike our approach in Section 5.4, we estimate the sub-pixel optical flow using the Lucas-Kanade algorithm [56] at the pixels detected by the FAST corner detector [75]. This allows us to accurately align the depth maps, which maximizes the number of pixels that are averaged and denoised. When the pose cannot be estimated, we use $\boldsymbol{R} = \boldsymbol{I}$ and $\boldsymbol{T} = \boldsymbol{0}$. This makes our approach similar to that proposed to Lin *et al.* [52]. However, in our experiments, this rarely happened because most scenes were sufficiently textured.

5.7.2 Reprojection

With the pose estimated, we then reproject the previous depth map, $\hat{D}_{t-1}(x, y)$, and the confidence map, $C_{t-1}(x, y)$. This will align the pixels of the previous depth map

(and confidence map) with that of the current depth map as well as account for the changes in depth between them.

To do so, we first obtain the 3D position of each pixel in the previous depth map. The 3D position of the i^{th} pixel located at (x_i, y_i), or $\boldsymbol{P_i}$, can be obtained using Eq. (2.2) and $\hat{D}_{t-1}(x, y)$. We then apply the pose, which we denote as $\boldsymbol{R}$ and $\boldsymbol{T}$, to each 3D point and obtain its new pixel coordinate, (x'_i, y'_i), and updated depth Z'_i, as follows:

$$(x'_i, y'_i)^T = \pi\left(\boldsymbol{RP_i} + \boldsymbol{T}\right) \qquad Z'_i = \hat{\boldsymbol{z}} \cdot \left(\boldsymbol{RP_i} + \boldsymbol{T}\right) \tag{5.9}$$

where $\pi(\cdot)$ is the projection operator as defined in Section 2.3.

This allows us to reproject both the depth map and confidence map, which we denoted as $\tilde{D}_{t-1}(x, y)$ and $\tilde{C}_{t-1}(x, y)$, respectively, where:

$$\tilde{D}_{t-1}(x'_i, y'_i) = Z'_i \qquad \tilde{C}_{t-1}(x'_i, y'_i) = C_{t-1}(x, y) \tag{5.10}$$

5.7.3 Weighted Average

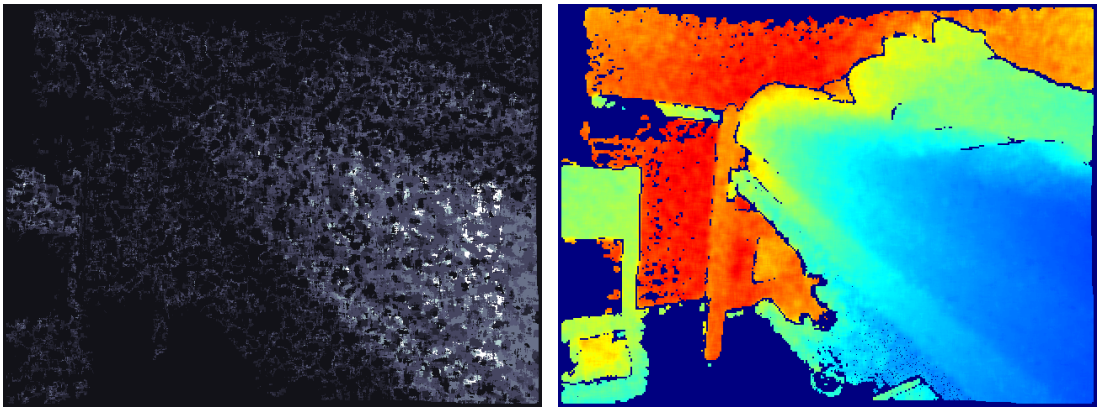
With the motion between the consecutive frames accounted for, we can now temporally average the depth maps to obtain $\hat{D}_t(x, y)$. We first apply a box filter to the current low power depth map, $D_t(x, y)$, to help reduce the noise in its pixels. We then perform a weighted average and weigh the pixels of the reprojected depth map, $\tilde{D}_{t-1}(x, y)$, using $\tilde{C}_{t-1}(x, y)$. For the i^{th} pixel, $\tilde{C}_{t-1}(x_i, y_i)$ stores the number of previous depth maps used to obtain the depth value, $\tilde{D}_{t-1}(x_i, y_i)$. Furthermore, when $\tilde{C}_{t-1}(x_i, y_i)$ is high, $\tilde{D}_{t-1}(x_i, y_i)$ can be thought of as a depth estimate from a *high power* depth map because it combines the data from many low power depth maps. With this intuition, we combine the depth maps as follows:

$$\hat{D}_t(x, y) = \begin{cases} \gamma \tilde{D}_{t-1}(x, y) + (1 - \gamma) D_t(x, y) & \text{if } |\tilde{D}_{t-1}(x, y) - D_t(x, y)| < \epsilon \\ D_t(x, y) & \text{otherwise} \end{cases} \tag{5.11}$$

where:

$$\gamma = \frac{\tilde{C}_{t-1}(x,y)}{\tilde{C}_{t-1}(x,y)+1} \tag{5.12}$$

and ϵ is a threshold to ensure that misalignments at depth boundaries (due to errors in the estimated pose) are not averaged. From Eq. (5.12), we see that when $C_{t-1}(x,y) >> 1$, we give more weight to the pixels in the previously filtered depth map. This is because those values are heavily filtered and less noisy than the current depth measurements. We visualize $C_{t-1}(x,y)$ alongside its corresponding depth map in Figure 5-9, where the brighter colors indicate the pixels in $\hat{D}_{t-1}(x,y)$ that have been recursively filtered using many previous depth maps.



$\boldsymbol{C_{t-1}(x,y)}$ $\boldsymbol{\hat{D}_{t-1}(x,y)}$

Figure 5-9: **Confidence Map**: We visualize the confidence map, where the brighter colors indicate the pixels in the depth map that have been recursively filtered using the data from many previous frames.

We also update the confidence map recursively. Denoting the updated confidence map as $C_t(x,y)$, we have:

$$C_t(x_i,y_i) = \begin{cases} 1 + C_{t-1}(x_i,y_i) & \text{if } |\tilde{D}_{t-1}(x,y) - D_t(x,y)| < \epsilon \\ 1 & \text{otherwise} \end{cases} \tag{5.13}$$

5.8 Part II: Evaluation

In this section, we evaluate the accuracy our approach using the data collected from a ToF camera and compare it to the previous approaches. We show that our algorithm

estimates accurate depth maps and outperforms the previous approaches.

5.8.1 Methodology

To evaluate our approach, we use a dataset of indoor scenes captured using a pulsed ToF camera [61]. This dataset contains 9 sequences, with frames that contain both a 640×480 IR image and a 640×480 depth map. Examples of these scenes are shown in Figure 5-10. To simulate the data obtained under various power settings, we add shot noise to these measurements. From Eq. (5.1), we see that the accumulated light for the i^{th} pixel, denoted as $S1_i$ and $S0_i$, can be written as:

$$S1_i = \frac{2Z_i\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2 + f^2}}{cf\tau} I_i \tag{5.14}$$

$$S0_i = I_i - S1_i \tag{5.15}$$

where Z_i is its depth taken from the depth map, I_i is its intensity taken from the IR image, (x_i, y_i) is its pixel coordinate, f is the principal distance, (x_c, y_c) is the principal point, τ is the pulse width of the emitted light, and c is the speed of light. In our experiments, we infer τ from the maximum range of the sensor [68].

Therefore, we can simulate depth maps obtained under different power settings by *scaling* I_i and sample measurements of $S0_i$ and $S1_i$ from a Poisson distribution with rates equal to the expressions in Eq. (5.14) and Eq. (5.15). As stated in [40], this models the noise that affects ToF cameras. This also gives us a ground truth depth map that can be used to evaluate our approach.

In our experiments, we apply our algorithm to 100 frames of each sequence for 10 different power settings. To evaluate the accuracy of our filtered depth maps, we use the percent mean relative error (MRE) as defined in Eq. (1.10). This is an appropriate metric for applications that need accurate depth in its immediate vicinity.

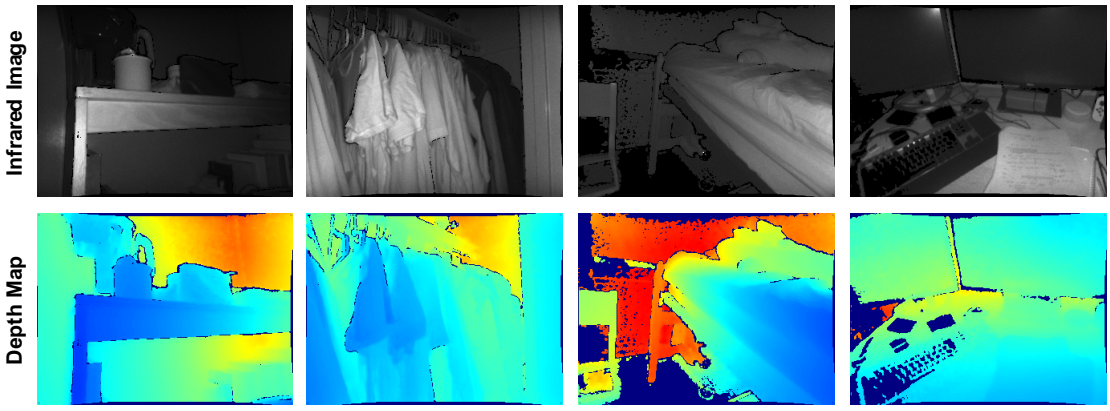


Figure 5-10: **Dataset**: We show examples of IR images and depth maps taken from the sequences used to evaluate our approach.

5.8.2 Implementation

We implement our algorithm on the ODROID XU-3 board [26], which is an embedded platform with an Exynos 5422 processor. This processor is used in the Samsung Galaxy S5 [78], and we evaluate our approach on this processor because it is representative of the computational resources available on mobile devices.

The resulting implementation filters depth maps in near real-time at 15.2 FPS for the data we evaluate on. Compared to the method in the first part of this chapter, this technique has the additional overhead of estimating corners using the FAST corner detector and estimating optical flow using the Lucas Kanade algorithm. Additionally, this algorithm also preprocesses the current depth map using a box filter as described in Section 5.7.3. These additional steps are required to ensure that our filtered depth maps are accurate.

5.8.3 Results

In Figure 5-11, we plot and compare the MRE of our filtered depth maps (denoted as *This Work*) to that of the unfiltered low power depth maps (denoted as *Low Power*). For each power setting (e.g., the scale factor as described in Section 5.8.1), we average the MRE of all of the depth maps for both *This Work* and *Low Power*. We also chose the parameters of our algorithm, namely ϵ, separately for each power setting to minimize this MRE. In this figure, our algorithm reduces the MRE of the

low power depth maps by up to 77%. As summarized in Table 5.3, when averaged across the different power settings, we see that we reduce the MRE of the low power depth maps from 10.45% to 2.97%. We show examples of our filtered depth maps in Figure 5-12 and provide an enlarged comparison of the low power depth and the filtered one in Figure 5-13.

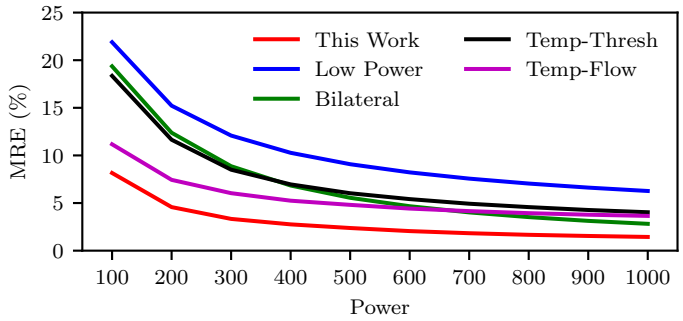


Figure 5-11: **Algorithm Performance Across Different Settings**: We compare the MRE of the various filtered depth maps to the unfiltered low power one across different power settings. We see that our approach achieves the lowest MRE and outperforms all of the different approaches.

	Mean	**Min**	**Max**
This Work	**2.97%**	**1.44%**	**8.16%**
Low Power	10.43%	6.27%	21.89%
Bilateral	7.11%	2.82%	19.36%
Temp-Thresh	7.47%	4.03%	18.35%
Temp-Flow	5.46%	3.65%	11.17%

Table 5.3: **MRE Statistics**: We present the mean, minimum, and maximum MRE across the different power settings for each of the different techniques. Our technique outperforms all of the competing approaches.

5.8.4 Comparison to Previous Approaches

We also compare our algorithm to approaches that are representative of those described in Section 5.6 as shown in Figure 5-11 and Table 5.3. For the spatial filtering techniques, we compare our approach to the bilateral filter (denoted as *Bilateral*), which is commonly used [48]. For the temporal filtering approaches, we compare our algorithm to the temporal filter proposed in [52] (denoted as *Temp-Thresh*), which

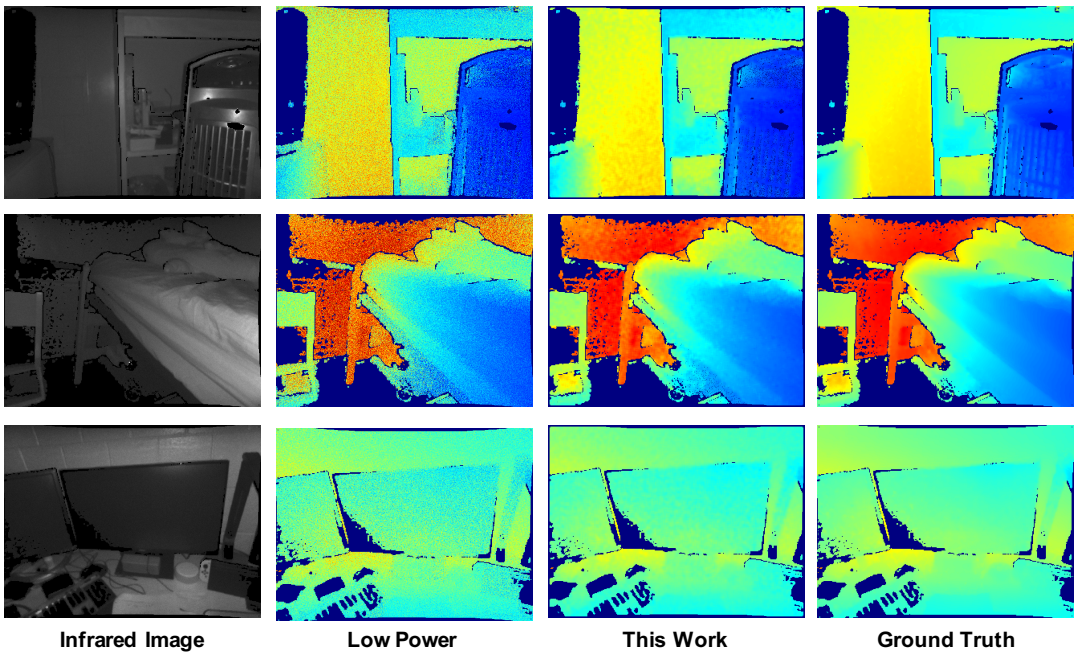


Figure 5-12: **Estimated Depth Maps**: We show examples of our filtered depth maps (denoted as *This Work*) and compare them to the low power and ground truth depth maps.

temporally averages the current pixel value with those taken from the same location in previous depth maps and are within a certain threshold of the current value. As stated in Section 5.7, our algorithm is similar to *Temp-Thresh* when the pose cannot be estimated. We also compare our approach to one that uses dense optical flow computed using the IR images to align and temporally filter multiple previous depth maps. This is similar to the approaches taken in [1, 22, 44, 57]. We denote this adapted technique as *Temp-Flow* and compute the optical flow using the IR images with the algorithm in [16]. For each of these algorithms, we choose the parameters that minimize the MRE for each power level.

Comparison to Spatial Filter

In Figure 5-11 and Table 5.3, we see that our algorithm outperforms the bilateral filter (*Bilateral*). Like other spatial filters, the bilateral filter assumes that the depth in a small region is constant and by averaging these pixels, the effects of noise can be mitigated. What makes the bilateral filter appealing is that it does not filter the

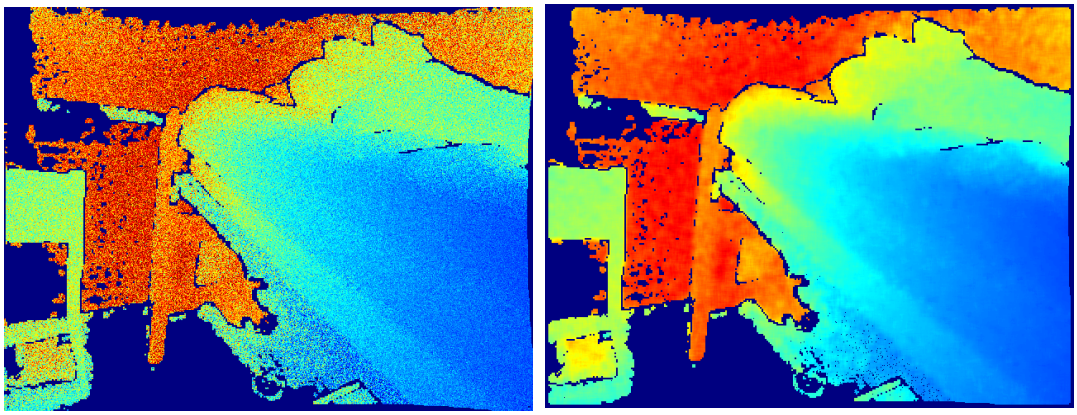


Figure 5-13: **Low Power Depth Map vs. This Work**: We show an enlarged example of the low power depth map before and after it is filtered (denoted as *This Work*).

pixel at *edges* (e.g., depth boundaries), which would introduce artifacts that make the filtered depth map appear blurry and inaccurate. The poor performance of the bilateral filter makes sense because the noise in the depth map is *non-stationary*. As shown in Eq. (5.3), the distribution of the noise *varies* from pixel to pixel depending on the intensity of the reflected light. This makes it difficult to set the filter parameters, in particular those of the spatial kernel, for the entire depth map because the noise distribution varies spatially. As a result, certain regions may appear to be edges even though they are not, and consequently, those pixels are not filtered.

In contrast, our approach does not take the local depth difference into account and instead directly averages the pixels of the reprojected and the current depth map at the same pixel coordinates to mitigate the noise. Because the reprojected and current depth maps are spatially aligned, these pixels have the same noise distribution. We also avoid artifacts at depth boundaries. Due to the noise in the low power depth map, we want to ensure that the depth maps are accurately aligned, which requires the pose to be estimated with sub-pixel optical flow at corners. As we stated in Section 5.8.2, this comes at the cost of increased latency. However, as we described in the first part of this chapter, computing the weights of a bilateral filter also has a non-trivial overhead. As shown in Table 5.4, our approach still has a higher frame

rate than that of the bilateral filter.

	Frame Rate (FPS)
This Work	**15.2**
Bilateral	8.6
Temp-Thresh	25.4
Temp-Flow	0.7

Table 5.4: **Frame Rate Comparison**: We profile the different approaches on an embedded processor with a Cortex-A15/Cortex-A7 octa core [26]. Due to our design choices, our approach offers the best tradeoff between the accuracy of the depth maps and the frame rate in which they are obtained.

Comparison to Temporal Filters

We also see that our approach outperforms the temporal filtering techniques, denoted as *Temp-Thresh* and *Temp-Flow* in Figure 5-11 and Table 5.3. As previously stated, *Temp-Thresh* simply averages the same pixel across multiple depth maps as long as they are within some of threshold of the current depth value. One advantage of this approach is its simplicity. As shown in Table 5.3, it filters depth maps with a higher frame rate than our approach. However, this comes at the cost of accuracy. Because the depth variance varies from pixel to pixel, it is challenging to select a threshold so that pixels with similar depth values are averaged. Furthermore, when there is motion between consecutive frames, fewer pixels are also averaged due to the changes in depth. Our algorithm does not suffer from these issues because it spatially align the depth maps and accounts for changes in depth.

This is also why our algorithms outperforms *Temp-Flow*, which aligns the previous depth maps with the current one by estimating dense optical flow between the IR images. *Temp-Flow* can only account for in-plane motion or small changes in depth, making it insufficient for the non-trivial motion in the dataset we evaluate on. Furthermore, computing accurate optical flow is also challenging. Because our approach uses the pose, which can be estimated robustly using sparse correspondences, it is less impacted by issues like uniform texture or the aperture effect. Another drawback of *Temp-Flow* is that computing dense optical flow is computationally expensive. As

shown in Table 5.3, we see that these techniques filter depth maps with a frame rate of 0.7 FPS, which is substantially lower than that of our approach.

Across all of these previous techniques, we see that our algorithm offers the best tradeoff between the accuracy of the filtered depth maps and the latency in which they are obtained.

5.8.5 Impact of Recursive Processing

We also compare our approach to a variant that is not recursive. To estimate depth maps, this algorithm, which we denote as *Multi-Frame*, stores multiple previous IR image and depth map pairs. To temporally filter a depth map, this approach estimates the pose between each of the previous depth maps and the current depth map, aligns them (as well as account for the changes in depth), and averages them all together. We compare the MRE of our approach to that of *Multi-Frame* in Figure 5-14. For each power setting, we use the optimal parameters, namely the number of previous frames to use, and compare its MRE to that of our recursive approach. Averaged across all of the power settings, we see that this variant has a MRE of 2.92%, which is slightly lower than the MRE of our approach (2.97%).

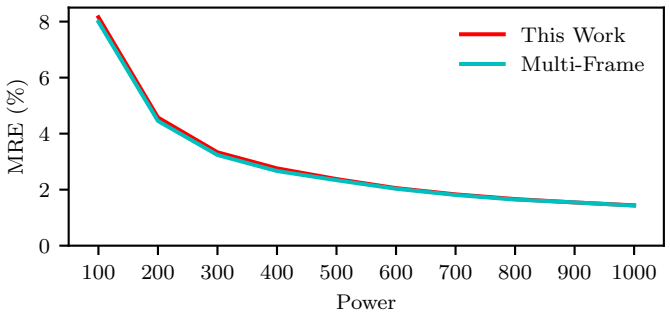


Figure 5-14: **Comparison to Multi-Frame Approach**: We compare our algorithm to a multi-frame variant that filters depth maps using a buffer of previous IR image and depth map pairs. We see that our recursive algorithm performs negligibly worse.

However, our approach uses less data. To quantify this, we define an *array* to be a 640×480 array of numbers. There are 3 types of arrays: images, depth maps, and confidence maps. For example, *This Work* stores three types of arrays (IR images, depth maps, and confidence maps), *Multi-Frame* and *Temp-Flow* store two types of

arrays (IR images and depth maps), and *Temp-Thresh* only stores one type (depth maps).

In Figure 5-15, we plot the total number of arrays that each approach has to store for the different power levels. For each power level, we searched for the minimal number that approximately minimizes the average MRE (e.g., we stop when more arrays only reduces the MRE negligibly) for *This Work*, *Multi-Frame*, *Temp-Thresh*, and *Temp-Flow*. We see that as the power increases, the total number of arrays generally decreases. This makes sense because the accuracy of the individual depth maps increases with power and any additional gain from temporal averaging is muted. Our approach also only needs to store three arrays in total (e.g., an IR image, a depth map, and a confidence map) which is lower than all of the other techniques. Because the other techniques are not recursive, they need to store multiple IR images and depth maps, which increases the total number of stored arrays. This is especially true for the lower power settings. Therefore, our approach also offers a beneficial tradeoff between the amount of data stored and the accuracy of the resulting depth map.

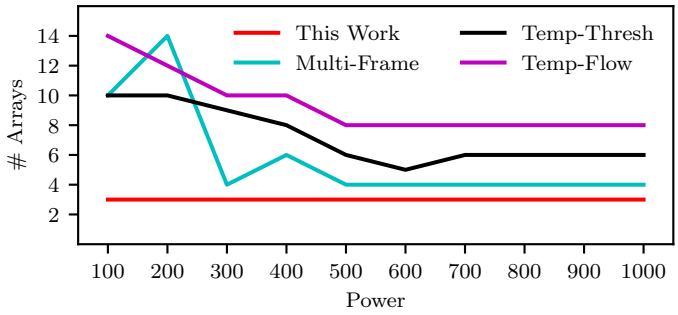


Figure 5-15: **Number of Arrays Stored**: We plot the minimum number of arrays that each algorithm needs to minimize the MRE for each power setting. Our recursive approach needs to store the least amount of data.

5.9 Summary

In this chapter, we address the challenge of lowering the power of ToF cameras using only the data from the sensor itself. To that end, we propose two approaches that

exploit the fact that the rigid motions in the scene can be efficiently estimated using the sparse optical flow between the IR images that a ToF camera acquires alongside the depth maps. First, we use this motion to adaptively vary the amount of light that the ToF camera emits. Instead of obtaining high power depth maps continuously, our approach obtains them at a lower frequency and uses the estimated motion to combine them with subsequent low power depth maps. This allows us to reduce the overall amount of emitted light by 81% while still obtaining accurate depth maps. We also consider the scenario where we only obtain low power depth maps. To denoise these depth maps, we then show that we can use the estimated rigid motion to recursively filter these depth maps efficiently. Compared to other similar approaches, our technique offers a better balance between the accuracy of the estimated depth map, the latency in which they are obtained, and the amount of data that must be stored. The resulting algorithm reduces the mean relative error of the low power depth maps by up to 77%, outperforming similar approaches. As a result of our design choices, both of these algorithms also run in up to real-time using the CPUs of an embedded processor.

Chapter 6

Conclusion and Future Work

In this thesis, we present different algorithms to lower the power of depth sensing for time-of-flight (ToF) cameras, which are appealing yet power hungry sensors. In this concluding chapter, we summarize our contributions and key insights and discuss future avenues of research.

6.1 Summary of Contributions and Key Insights

To lower the power for depth sensing using ToF cameras, our algorithms leverage multimodal data, temporal relationships, and the assumption of rigidity to obtain accurate depth maps efficiently. When RGB images are concurrently collected, we use them to reduce the usage of the ToF camera and to estimate new depth maps instead. When only using the data from a ToF camera, we use its infrared images to reduce the power required to obtain accurate depth maps. We demonstrate the accuracy of our algorithms using a variety of different datasets and show that our depth maps can be used in a real augmented reality application. We also show that our algorithms are efficient and can obtain depth maps with low latency by evaluating them on both a standard laptop computer and an embedded processor [26]. In all of our approaches, we leverage the following insights.

The World Is Nearly Rigid We find that leveraging rigidity allows us to efficiently obtain accurate depth maps. When RGB images are concurrently collected, we use their sparse pixel-wise motions to estimate the rigid motions in the scene. As a result, we are able to obtain new depth maps with a mean relative error (MRE) of 1% even when we reduce the usage of the ToF camera by 85%. For a dynamic scene, we find that by modeling its motion to be composed of multiple independent rigid motions, we are able to estimate new depth maps with a MRE of 2.5% even when we reduce the usage of the ToF camera by 90%. When only using the data from a ToF camera, we use the pixel-wise motion of its infrared images to estimate the rigid motions and use them to temporally combine consecutive depth maps. This is essential for our scheme that adaptively varies the amount of emitted light to infrequently obtain high power depth maps and use them to denoise subsequent low power ones. Using the rigid motion to account for the changes in depth between the high power depth maps and the subsequent low power ones allows us to combine them and reduce the mean relative error of the low power depth maps by up to 64% and the amount of emitted light by 81%. The same is true for our recursive filtering approach, where we reduce the mean relative error of the low power depth maps by 77%.

Sparse Processing Trumps Dense Operations Another benefit of the rigidity assumption is that it allows us to use sparse operations, in particular sparse optical flow, to reduce computation. Because the rigid motion can be estimated using sparse optical flow, we are able to avoid estimating the dense optical flow, unlike many approaches in the literature. The use of sparse optical flow algorithms allows our techniques to run in up to real-time on both standard and embedded computing platforms. In the case of rigid objects and scenes, we are even able to lower the overall system power for depth sensing because our algorithms can estimate depth maps in real-time on the low power cores of our embedded processor [26]. For our recursive filtering approach, we use the rigid motion obtained through the sparse operations to directly compute the dense correspondences between the consecutive depth maps instead of estimating the dense optical flow.

Using Previous Depth Reduces Computational Complexity Another way of looking at our strategies is that by adding a small amount of depth into the problem formulation (e.g., that of depth map estimation), we can reduce the computation required to solve an otherwise computationally expensive problem. This is especially true when we estimate depth in dynamic scenes, where we use the previous depth map to compute the photometric error to guide the assignment of the estimated rigid motions. This allows us to avoid object-level segmentation used by similar approaches.

6.2 Future Directions

There are many exciting ways to extend our research. We describe some directions below.

Towards a Real-Time and Low Power Implementation One way to increase throughput and reduce the overall power for depth sensing is to implement our algorithms in hardware. By designing energy efficient accelerators for our approach, our algorithms can be directly deployed alongside ToF cameras to reduce the overall system power for depth sensing.

Infilling Missing Depth When a previously occluded region is uncovered, its depth is missing in consecutively estimated depth maps. One way to potentially fill this in is to leverage ideas from structure-from-motion and exploit concurrently collected images to triangulate the depth in these missing regions. The research question would be centered on how to do this accurately and efficiently, especially for small baselines.

Depth Map Super-Resolution Typically, the resolution of depth maps lag behind that of images. One way to resolve this is to apply multi-frame super-resolution techniques to increase the resolution of the depth map. For rigid scenes and environments, we may be able to use the estimated rigid motion to align the depth maps to obtain a higher resolution depth map in an efficient manner.

Bibliography

[1] Razmik Avetisyan, Christian Rosenke, Martin Luboschik, and Oliver Staadt. Temporal Filtering of Depth Images Using Optical Flow. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 271–275, 2016.

[2] Cyrus S. Bamji, Swati Mehta, Barry Thompson, Tamer Elkhatib, Stefan Wurster, Onur Akkaya, Andrew Payne, John Godbaz, Mike Fenton, Vijay Rajasekaran, Larry Prather, Satya Nagaraja, Vishali Mogallapu, Dane Snow, Rich McCauley, Mustansir Mukadam, Iskender Agi, Shaun McCarthy, Zhanping Xu, Travis Perry, William Qian, Vei-Han Chan, Prabhu Adepu, Gazi Ali, Muneeb Ahmed, Aditya Mukherjee, Sheethal Nayak, Dave Gampell, Sunil Acharya, Lou Kordus, and Pat O'Connor. IMpixel 65nm BSI 320MHz Demodulated TOF Image Sensor With 3μm Global Shutter Pixels and Analog Binning. In *International Solid-State Circuits Conference*, pages 94–96. IEEE, 2018.

[3] Basler. Basler Time-of-Flight.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[5] Simone Bianco, Gianluigi Ciocca, and Davide Marelli. Evaluating the Performance of Structure from Motion Pipelines. *Journal of Imaging*, 4(8):98, 2018.

[6] Andreas Breitbarth, Timothy Schardt, Cosima Kind, Julia Brinkmann, Paul-Gerald Dittrich, and Gunther Notni. Measurement Accuracy and Dependence on External Influences of the iPhone X TrueDepth Sensor. In Maik Rosenberger, Paul-Gerald Dittrich, and Bernhard Zagar, editors, *Photonics and Education in Measurement Science 2019*, volume 11144, pages 27–33. International Society for Optics and Photonics, SPIE, 2019.

[7] D J Butler, J Wulff, G B Stanley, and M J Black. A Naturalistic Open Source Movie for Optical Flow Evaluation. In *European Conference on Computer Vision*, 2012.

[8] P. Buyssens, M. Daisy, D. Tschumperle, and O. Lezoray. Superpixel-Based Depth Map Inpainting for RGB-D View Synthesis. In *International Conference on Image Processing*. IEEE, 2015.

[9] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth Prediction Without the Sensors: Leveraging Structure for Unsupervised Learning from Monocular Videos. In *Conference on Artificial Intelligence*. AAAI, 2018.

[10] Derek Chan, Hylke Buisman, Christian Theobalt, and Sebastian Thrun. A Noise-Aware Filter for Real-Time Depth Upsampling. In *Workshop on Multi-Camera and Multi-Modal Sensor Fusion Algorithms and Applications*, 2008.

[11] Jinwook Choi, Dongbo Min, and Kwanghoon Sohn. 2D-Plus-Depth Based Resolution and Frame-Rate Up-Conversion Technique for Depth Video. *IEEE Transactions on Consumer Electronics*, 56(4):2489–2497, 2010.

[12] Andrea Colaco, Ahmed Kirmani, Nan-wei Gong, Tim Mcgarry, Laurence Watkins, and Vivek K Goyal. 3dim : Compact and Low Power Time-of-Flight Sensor for 3D Capture Using Parametric Signal Processing. *International Image Sensor Workshop*, 2013.

[13] Analog Devices. AD-96TOF1-EBZ: 3D Time of Flight Development Platform.

[14] J Engel, T Schöps, and D Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *European Conference on Computer Vision*, 2014.

[15] EveryMac. Apple MacBook Pro 13-Inch "Core i5" 2.7 Early 2015 Specs.

[16] Gunnar Farneback. Two-Frame Motion Estimation Based on Polynomial Expansion. *Scandinavian Conference on Image Analysis*, 2749(1):363–370, 2003.

[17] Juan-Antonio Fernández-Madrigal and José Luis Blanco Claraco. *Simultaneous Localization and Mapping for Mobile Robots*. Advances in Computational Intelligence and Robotics. IGI Global, 2013.

[18] Martin A Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting with. *Communications of the ACM*, 24:381–395, 1981.

[19] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical Flow Modeling and Computation: A Survey. *Computer Vision and Image Understanding*, 134:1–21, 2015.

[20] Mario Frank, Matthias Plaue, and Fred A. Hamprecht. Denoising of Continuous-Wave Time-of-Flight Depth Images Using Confidence Measures. *Optical Engineering*, 48(7):077003, 2009.

[21] A Gaidon, Q Wang, Y Cabon, and E Vig. Virtual Worlds as Proxy for Multi-Object Tracking Analysis. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[22] Mihail Georgiev, Atanas Gotchev, and Miska Hannuksela. Real-Time Denoising of ToF Measurements by Spatio-Temporal Non-Local Mean Filtering. In *International Conference on Multimedia and Expo Workshops*. IEEE, 2013.

[23] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. Depth from Videos in the Wild: Unsupervised Monocular Depth Learning from Unknown Cameras. In *International Conference on Computer Vision*, 2019.

[24] A Handa, T Whelan, J B McDonald, and A J Davison. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *International Conference on Robotics and Automation*. IEEE, 2014.

[25] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Horaud. *Time-of-Flight Cameras*. SpringerBriefs in Computer Science. Springer London, London, 2013.

[26] HardKernel. ODROID-XU3.

[27] C Harris and M Stephens. A Combined Corner and Edge Detector. In *Alvey Vision Conference*, pages 147–151, 1988.

[28] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[29] Kaiming He, Jian Sun, and Xiaoou Tang. Guided Image Filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, 2013.

[30] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. *International Journal of Robotics Research*, 31(5):647–663, 2010.

[31] José-Juan Hernández-López, Ana-Linnet Quintanilla-Olvera, José-Luis López-Ramírez, Francisco-Javier Rangel-Butanda, Mario-Alberto Ibarra-Manzano, and Dora-Luz Almanza-Ojeda. Detecting Objects Using Color and Depth Segmentation with Kinect Sensor. *Procedia Technology*, 3:196–204, 2012.

[32] Daniel Herrera C., Juho Kannala, L'ubor Ladický, and Janne Heikkilä. Depth Map Inpainting Under a Second-Order Smoothness Prior. In *Image Analysis*, pages 555–566. Springer Berlin Heidelberg, 2013.

[33] Berthold K.P. Horn. Recovering baseline and orientation from essential matrix.

[34] Berthold K.P. Horn. *Robot Vision*. MIT Electrical Engineering and Computer Science Series. MIT Press, 1986.

[35] Berthold K.P. Horn. Motion Fields Are Hardly Ever Ambiguous. *International Journal of Computer Vision*, 1(3):259–274, 1988.

[36] Berthold K.P. Horn. Relative Orientation. *International Journal of Computer Vision*, 4(1):59–78, 1990.

[37] Berthold K.P. Horn and Brian G Schunck. Determining Optical Flow. Technical report, USA, 1980.

[38] Benjamin Huhle, Timo Schairer, Philipp Jenke, and Wolfgang Strasser. Robust Non-Local Denoising of Colored Depth Data. In *Conference on Computer Vision and Pattern Recognition Workshop*, pages 1–7. IEEE, 2008.

[39] IFM. IFM O3D302.

[40] Julio Illade-Quinteiro, Víctor Brea, Paula López, Diego Cabello, and Gines Doménech-Asensi. Distance Measurement Error in Time-of-Flight Sensors Due to Shot Noise. *Sensors*, 15(3):4624–4642, feb 2015.

[41] Wonkwi Jang, Changsoo Je, Yongduek Seo, and Sang Wook Lee. Structured-Light Stereo: Comparative Analysis and Integration of Structured-Light and Active Stereo for Measuring Dynamic Shape. *Optics and Lasers in Engineering*, 51(11):1255–1264, nov 2013.

[42] Sebastian Hoppe Nesgaard Jensen, Alessio Del Bue, Mads Emil Brix Doest, and Henrik Aanæs. A Benchmark and Evaluation of Non-Rigid Structure from Motion. jan 2018.

[43] Kevin Karsch, Ce Liu, and Sing Bing Kang. Depth Transfer: Depth Extraction from Video Using Non-Parametric Sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2144–2158, 2014.

[44] Sung-Yeol Kim, Ji-Ho Cho, Andreas Koschan, and Mongi A. Abidi. Spatial and Temporal Enhancement of Depth Images Captured by a Time-of-Flight Depth Sensor. In *International Conference on Pattern Recognition*. IEEE, 2010.

[45] T Koga, K Iinuma, A Hirano, Y Iijima, and T Ishiguro. Motion Compensated Interframe Coding for Video Conferencing. In *NTC81*, 1981.

[46] Suryansh Kumar, Yuchao Dai, and Hongdong Li. Monocular Dense 3D Reconstruction of a Complex Dynamic Scene from Two Perspective Frames. In *International Conference on Computer Vision*. IEEE, 2017.

[47] Suryansh Kumar, Ram Srivatsav Ghorakavi, Yuchao Dai, and Hongdong Li. Dense Depth Estimation of a Complex Dynamic Scene without Explicit 3D Motion Estimation. feb 2019.

[48] Frank Lenzen, Kwang In Kim, Henrik Schäfer, Rahul Nair, Stephan Meister, Florian Becker, and Christoph S Garbe. Denoising Strategies for Time-of-Flight Data. In Marcin Grzegorzek, Christian Theobalt, Andreas Kolb, Christian Theobalt, and Reinhard Koch, editors, *Time-of-Flight and Depth Imaging: Sensors, Algorithms, and Applications*, volume 8200, pages 25–45. Springer, 2013.

[49] Frank Lenzen, Henrik Schäfer, and Christoph Garbe. Denoising Time-Of-Flight Data with Adaptive Total Variation. pages 337–346. 2011.

[50] Yanjie Li, Lifeng Sun, and Tianfan Xue. Fast Frame-Rate Up-Conversion of Depth Video Via Video Coding. In *International Conference on Multimedia*, page 1317. ACM, 2011.

[51] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T. Freeman. Learning the Depths of Moving People by Watching Frozen People. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[52] Bor-Shing Lin, Mei-Ju Su, Po-Hsun Cheng, Po-Jui Tseng, and Sao-Jie Chen. Temporal and Spatial Denoising of Depth Maps. *Sensors*, 15(8):18506–18525, 2015.

[53] Junyi Liu, Xiaojin Gong, and Jilin Liu. Guided Inpainting and Filtering for Kinect Depth Maps. In *International Conference on Pattern Recognition*, 2012.

[54] Hugh Christopher Longuet-Higgins. A Computer Algorithm for Reconstructing a Scene from Two Projections. *Nature*, 293(5828):133–135, 1981.

[55] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[56] Bruce D Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *International Joint Conference on Artificial Intelligence*, 1981.

[57] Sergey Matyunin, Dmitriy Vatolin, Yury Berdnikov, and Maxim Smirnov. Temporal Filtering for Depth Maps Generated by Kinect Depth Camera. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*. IEEE, 2011.

[58] Dan Miao, Jingjing Fu, Yan Lu, Shipeng Li, and Chang Wen Chen. Texture-Assisted Kinect Depth Inpainting. In *International Symposium on Circuits and Systems*. IEEE, 2012.

[59] Raul Mur-Artal and Juan D. Tardos. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, oct 2017.

[60] David Nister. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[61] James Noraky, Charles Mathy, Alan Cheng, and Vivienne Sze. Low Power Adaptive Time-Of-Flight Imaging For Multiple Rigid Objects. In *International Conference on Image Processing*. IEEE, 2019.

[62] James Noraky and Vivienne Sze. Low Power Depth Estimation for Time-of-Flight Imaging. In *International Conference on Image Processing*, pages 2114–2118. IEEE, 2017.

[63] James Noraky and Vivienne Sze. Depth Estimation of Non-Rigid Objects for Time-Of-Flight Imaging. In *International Conference on Image Processing*. IEEE, 2018.

[64] James Noraky and Vivienne Sze. Low Power Depth Estimation of Rigid Objects for Time-of-Flight Imaging. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[65] James Noraky and Vivienne Sze. Depth Map Estimation of Dynamic Scenes Using Prior Depth Information. feb 2020.

[66] ODOS Imaging. Swift-E.

[67] Jaesik Park, Hyeongwoo Kim, Yu-Wing Tai, Michael S. Brown, and Inso Kweon. High Quality Depth Map Upsampling for 3D-TOF Cameras. In *International Conference on Computer Vision*. IEEE, 2011.

[68] PicoVR. Pico Zense DCAM710.

[69] Andry Maykol Pinto, Paulo Costa, Antonio P. Moreira, Luis F. Rocha, Germano Veiga, and Eduardo Moreira. Evaluation of Depth Sensors for Robotic Applications. In *IEEE International Conference on Autonomous Robot Systems and Competitions*, pages 139–143. IEEE, apr 2015.

[70] PMD. CamBoard Pico Flexx.

[71] PMD. CamBoard Pico Monstar.

[72] Fei Qi, Junyu Han, Pengjin Wang, Guangming Shi, and Fu Li. Structure Guided Fusion for Depth Map Inpainting. *Pattern Recognition Letters*, 34(1):70–76, 2013.

[73] Rene Ranftl, Vibhav Vineet, Qifeng Chen, and Vladlen Koltun. Dense Monocular Depth Estimation in Complex Dynamic Scenes. In *Conference on Computer Vision and Pattern Recognition*, pages 4058–4066. IEEE, 2016.

[74] Christoph Rhemann, Asmaa Hosni, Michael Bleyer, Carsten Rother, and Margrit Gelautz. Fast Cost-Volume Filtering for Visual Correspondence and Beyond. In *Conference on Computer Vision and Pattern Recognition*. IEEE, 2011.

[75] E. Rosten and T. Drummond. Fusing Points and Lines for High Performance Tracking. In *International Conference on Computer Vision*. IEEE, 2005.

[76] Anastasios Roussos, Chris Russell, Ravi Garg, and Lourdes Agapito. Dense Multibody Motion Estimation and Reconstruction From a Handheld Camera. In *International Symposium on Mixed and Augmented Reality*. IEEE, 2012.

[77] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571. IEEE, 2011.

[78] Samsung. Exynos 5 Octa 5422 Processor.

[79] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1):7–42, 2002.

[80] Adam Schmidt, Michał Fularz, Marek Kraft, Andrzej Kasiński, and Michał Nowicki. An Indoor RGB-D Dataset for the Evaluation of Robot Navigation Algorithms. In *Lecture Notes in Computer Science*, volume 8192 LNCS, pages 321–329. 2013.

[81] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition*, pages 4104–4113. IEEE, 2016.

[82] Alexander Seitel, Thiago R. dos Santos, Sven Mersmann, Jochen Penne, Anja Groch, Kwong Yung, Ralf Tetzlaff, Hans-Peter Meinzer, and Lena Maier-Hein. Adaptive Bilateral Filter for Image Denoising and Its Application to In-Vitro Time-of-Flight Data. page 796423, mar 2011.

[83] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *European Conference on Computer Vision*, 2012.

[84] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo Tourism. In *ACM SIGGRAPH*, page 835. ACM Press, 2006.

[85] Jurgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *International Conference on Intelligent Robots and Systems*, 2012.

[86] Aydin Varol, Mathieu Salzmann, Pascal Fua, and Raquel Urtasun. A Constrained Latent Variable Model. In *Conference on Computer Vision and Pattern Recognition*, 2012.

[87] Aydin Varol, Mathieu Salzmann, Engin Tola, and Pascal Fua. Template-free monocular reconstruction of deformable surfaces. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1811–1818. IEEE, sep 2009.

[88] Daniel Wagner. Depth Cameras for Mobile AR: From IPhones to Wearables and Beyond, 2018.

[89] Hung Ming Wang, Chun Hao Huang, and Jar Ferr Yang. Depth Maps Interpolation from Existing Pairs of Keyframes and Depth Maps for 3D Video Generation. In *International Symposium on Circuits and Systems*. IEEE, 2010.

[90] Oliver Wasenmuller, Marcel Meyer, and Didier Stricker. CoRBS: Comprehensive RGB-D benchmark for SLAM Using Kinect V2. In *Winter Conference on Applications of Computer Vision*, pages 1–7. IEEE, 2016.

[91] Changchang Wu. Towards Linear-Time Incremental Structure from Motion. In *International Conference on 3D Vision*, pages 127–134. IEEE, 2013.

[92] Rui Yao, Guosheng Lin, Shixiong Xia, Jiaqi Zhao, and Yong Zhou. Video Object Segmentation and Tracking: A Survey. 2019.

[93] Donghoon Yeo, Ehsan Ul haq, Jongdae Kim, Mirza Waqar Baig, and Hyunchul Shin. Adaptive Bilateral Filtering for Noise Removal in Depth Upsampling. In *International SoC Design Conference*, pages 36–39. IEEE, 2010.

[94] Guofeng Zhang, Jiaya Jia, and Hujun Bao. Simultaneous Multi-Body Stereo and Segmentation. In *International Conference on Computer Vision*. IEEE, 2011.

[95] Yongbing Zhang, Jian Zhang, and Qionghai Dai. Texture Aided Depth Frame Interpolation. *Signal Processing: Image Communication*, 29(8):864–874, 2014.