ABSTRACT

Since its release, Kinect has been the de facto standard for low-cost RGB-D sensors. An infrared laser ray shot through an holographic diffraction grating projects a fixed dot pattern which is captured using an infrared camera. The pseudo-random pattern ensures that a simple block matching algorithm suffices to provide reliable depth estimates, allowing a cost-effective implementation. In this paper, we analyze the software limitations of Kinect’s method, which allows us to propose algorithms that provide better precision. First, we analyze the dot pattern: we measure its pincushion distortion and its effect on the dot density, which is smaller towards the edges of the image. Then, we analyze the behavior of Block Matching algorithms, we show how Kinect’s Block Matching implementation is; in general; limited by the dot density of the pattern, and a significant spatial bias is introduced as a result. We propose an efficient approach to estimate the disparity of each dot, allowing us to produce a point cloud with better spatial resolution than Block Matching algorithms.

Index Terms—Kinect, RGB-D, pattern, bias

1. INTRODUCTION

Microsoft Kinect needs little introduction. It was the first affordable camera that provided reliable depth maps in indoor scenarios and it has become widely popular both in consumer and research areas.

Kinect itself is a combination of three different USB devices: a microphone array, a tilt motor and an RGB-D camera. The RGB-D device is powered by PrimeSense technology, which also powers Wavi Xtion family of sensors from ASUS.

Kinect uses a technology called texture projection where a pattern is projected to the scene in order to simplify the correspondence problem by providing unique texture patches. The common approach used in texture projection systems is to use a standard LCD projector to cast the texture over the scene and a stereo camera to obtain the disparity map, however by using a fixed texture, a simpler projection component can be used as in the PR2 sensor head [1]. One of the simplest options is to use an infrared(IR) laser coupled with a fiber grating device to project a grid of points [2], although inexpensive, the resulting dot pattern is regular and narrowly limits the depth range and spatial resolution achievable. Kinect solved this problem by using a two layered fiber grating with an holographic structure that projects a pseudo-random dot pattern [3]. As the projected pattern is fixed and known, a single infrared camera suffices to triangulate accurately the texture, providing additional cost savings over stereo camera systems. Finally, as Block Matching is a hardware friendly technique [4], Kinect implements it to provide a disparity map from the IR image. The conversion from disparity to depth is not performed by the device.

Being a closed system, several groups have analyzed Kinect from different point of views. From the hardware, communication and capabilities side the work from Freenect has been crucial [5]. Several papers have analyzed Kinect and
have compared it against other depth range alternatives [6–9],
those helped to identify the bias that Kinect systems show
against temperature changes. Finally some groups have ana-
lyzed the pattern and the optical characteristics [10]. In our
group, we created a model able to predict the depth map given
a source infrared image taken by the same Kinect [11]. In this
paper we analyze our Kinect model in order to better deter-
mine its limitations and provide better algorithms to generate
the depth map.

We start by analyzing the limitations of the projected dot
pattern. As the IR camera does not obtain any information
from the dark parts of the image, the depth information is
available only from the parts of the scene illuminated by a
dot. Therefore, the dot distribution fixes the actual maximum
spatial resolution achievable by Kinect. Our analysis show
that an circular object must have a radius of 3.25 pixels in
order to be recognized by Kinect.

Then, we analyze the behavior of alternate Block Match-
ing(BM) algorithms on the system Fig. 1. We have imple-
mented a Kinect simulator using the model suggested by [11]
to evaluate depth algorithms. It allows us to simulate artificial
scenarios and thus compare our results to a known ground
truth. We found that although it is possible to improve the
depth resolution with better BM algorithms, the spatial reso-
nution remains poor.

Finally we propose to estimate the depth of each individu-
al projected dot. This approach allows us to provide a 28K
point cloud, close to the actual resolution limit of the pro-
jected pattern. Although the grid structure is lost, this ap-
proach is be useful to projects that use point cloud informa-
tion in 3D space. The fact that only 28K points are provided
increases the performance of the system with respect to the
original BM algorithm with 300K points.

2. KINECT MODEL

We developed a Kinect simulator based on the model pre-
sented in [11] to analyze the properties of the suggested depth
algorithms. Given a triangle mesh based scene definition,
we use raytracing to simulate the infrared view that Kinect
would perceive. The simulator provides ground truth of the
depth, and simulates the block matching algorithm performed
by Kinect.

First, we obtain the calibration pattern of a Kinect by re-
viering its model:

\[ D(x, y) = \arg \min_k |I(x + k, y) - R(x, y)| \]

where \( D \) is the disparity provided by Kinect, \( I \) is the IR image
and \( R \) is the reference pattern.

\[ R'(x, y) = I(x + D(x, y), y) . \]

Therefore, if we obtain simultaneously the IR image \( I \) and the
disparity field \( D \), we can approximate the reference image \( R' \)
by displacing the image from \( I \) the amount noted by \( D \).

Although the exact block size used in Kinect is unknown,
results are consistent with a 16x16 block size [11].

From the OpenNI driver, we know that the reference im-
age is a calibration image taken at \( p = 1200mm \) from the
camera with a 100 pixel bias. Knowing the focal length \( f_x =
580 \) (at VGA resolution) and the baseline \( b = 75mm \). And the
disparity is provided at a resolution of 8x of the VGA out-
put, the depth \( d \) can be calculated as:

\[ d = b \times \frac{f_x}{100 + b \times f_x/p - (1/8) \times disp} . \] (3)

Is it important to notice that Kinect provides depth at
VGA resolution, however internally processes IR images at
SXGA resolution. In this paper we express pixel sizes at
VGA resolution unless otherwise noted.

3. ANALYSIS OF THE DOT PATTERN

To analyze the pattern, we use a HSXGA version of the refer-
ence image to localize and annotate each individual dot with
high precision.

As the pattern shows a strong vignetting effect, Kinect
chooses a brightness setting that causes saturation on the dots
close to the center of the image. It is not possible to alter
the brightness setting, so we apply a Gaussian filtering with
\( \sigma = 3px \) to allow us to find dots as local maxima. A threshold
based on the mean brightness over its neighborhood is used to
eliminate noise and small sparkles that appear as side effects
of the holographic grating. Finally we use the 3x3 neighbor-
hood around the maxima to estimate its peak with subpixel
precision. A total of 28117 dots are detected.

Assuming a block size of 8x8 pixels (on VGA), we an-
alyze the density of the dots/block, the block bias, and the
resolution (see Fig. 2).

3.1. Dot density

We measure the density of the dots by counting, per each
pixel on the VGA image, how many dots would be included
in its 8x8 neighborhood. The top value is 14, and the mini-
um value is 1, however the mean value is 7.62 ± 1.23
dots/block on the center of the pattern and drops to 3.07 ±
1.18 dots/block at the edges.

3.2. Block bias

We define the block bias as the mean position of the dots
included within a block with respect to the its center. This
measure provides us with an indication of the precision of
the Kinect spatial information. If we approximate the local
neighborhood of a pixel as a flat surface, and the output of
the BM algorithm as the mean disparity of each individual
dot in the block, then the distance reported will be that of the
mean position the dots within the block, which may not be
the actual center of the block (i.e. the pixel we are testing). This is the source of the non-sharp edges present in the depth field (see Fig. 1). The block bias at the center of the pattern is 0.64 ±0.34 pixels and 1.60 ±1.02 pixels at the edges. In fact, less than 40% of the pixels in the center of the pattern present a bias smaller than 0.5 pixels, and this number drops to less than 10% at the edges. This means that, in ideal conditions, most pixels report a disparity value that would have been better reported by a neighboring pixel.

3.3. Resolution

As the pattern is non uniform, we define its spatial resolution as the capability to detect a circular feature in the scene.

First we would test an ideal scenario: we assume that we have an oracle that allows us to perfectly solve the correspondence problem and, in consequence, our circular test object only requires to be be illuminated by a single dot of the pattern to be detected. For each pixel, the distance to its closest dot will indicate the minimum detectable radius, and ranges from 1.18 ±0.50 to 1.98 ±0.89 pixels.

However, if an object is in front of a background of similar albedo, it must cover more than half of the dots of a block to be detected. In this case, the dot density plays a smaller role, and the minimum detectable radius for a circular object is fairly constant (from 3.24 ±0.44 to 3.38 ±0.75 pixels).

4. BLOCK MATCHING ALGORITHMS

Kinect’s Block Matching algorithm uses a fixed point resolution corresponding to 1/8th of a VGA pixel.

As the IR image can be captured at a SXGA, we have evaluated floating point block matching algorithms directly applied to the higher resolution IR image expecting better depth resolution.

We have evaluated Konolige’s optimized BM implementations from OpenCV [12], first on SXGA resolution and later on QSXGA (2x2 SXGA).

SXGA takes 220ms to process an image and obtains a similar performance that of Kinect’s algorithm, while BM on QSXGA takes more than 12 seconds to process a single image although obtains a significantly better performance (see Fig. 3).

Fig. 3: Mean disparity error on the central part of a tilted plane in pixels. Solid: Kinect’s algorithm. Dotted: BM at SXGA resolution. Dash-dotted: BM at QSXGA resolution.
5. DOT-BASED POINT CLOUD

We have already discussed how Kinect can only obtain depth information from the dots of the projected pattern. Ideally, a system that assigned a disparity value to each visible dot would be optimal, however the problem is hard as the dots are indistinguishable between them.

We suggest to use a simple BM algorithm to solve the correspondence problem. This provides us a gross estimate of the pattern dots’ position within the IR image, which will be refined.

As a prerequisite, we localize with precision the dots from the reference image and create a table that lists, for each pixel of the reference image, all the dots in its 8x8 neighborhood. The online algorithm is as follows:

1. Apply BM between the IR and the reference image.

2. For each IR pixel \((i, j)\) with disparity \(d\), calculate its counterpart on the reference image \(R'(i, j + d)\), and add \(d\) as a possible disparity to all pattern dots in its 8x8 neighborhood.

3. For each dot in the pattern, cluster all disparity candidates (mean-shift \(\sigma = 1\)). Each disparity candidate is projected back to the IR image and the brightest one is selected.

Unlike Kinect’s, this algorithm does not output a regular grid. Instead it provides a 3D point cloud where the \(z\) coordinate is calculated from the disparity between a known dot in the pattern and its localization in the IR image. Then the \(x\) and \(y\) coordinates are calculated from the localization of the dot in the IR image and its depth.

We have evaluated this algorithm on synthetic and realistic scenarios (see Fig. 4) and found a performance of around 1.75 frames per second\(^1\). In our C++ implementation, the BM has a fixed cost of 290ms, pixel processing takes 50ms, mean-shift clustering 220ms and the final sub-pixel estimation 100ms. However the algorithm has ample margin for parallelization and could be implemented in a GPU for increased performance.

This algorithm achieves better depth precision than block matching algorithms (see Fig. 5).

6. CONCLUSIONS AND FUTURE WORK

We have analyzed Kinect’s Block Matching algorithm quantifying important characteristics such as the minimum size of a detectable feature, and the spatial bias induced by the unbalanced distribution of dots within a block. Using this analysis we propose a better Block Matching algorithm that improves depth precision. However, Block Matching algorithms provide blurred depth maps and do not compensate for the spatial bias. Therefore, we have proposed an efficient method to provide a sparse point cloud by estimating the disparity of each individual dot in the projected pattern. This method avoids unnecessary interpolation, therefore it provides an unbiased point cloud with higher spatial resolution than the original algorithm using only one tenth of the 3d points.

\(^1\) Intel(R) i5 760 @ 2.80GHz. Code available at: http://cvhci.anthropomatik.kit.edu/~manel/kinect
7. REFERENCES


