IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 January 2023, accepted 14 February 2023, date of publication 23 February 2023, date of current version 28 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3248070

RESEARCH ARTICLE

Model Based Design of a Real Time FPGA-Based Lens Undistortion and Image Rectification Algorithm for Stereo Imaging

DANIEL S. KAPUTA, (Member, IEEE), AND KRYSTIAN A. DERHAK
Rochester Institute of Technology, Rochester, NY 14623, USA
Corresponding author: Daniel S. Kaputa (dskiee@rit.edu)

ABSTRACT Low cost imaging sensors and powerful embedded computers have taken the field of computer vision to new heights. One of the challenges that remains is to shorten the development time that it takes to target one's algorithm to hardware. This work details the steps necessary to use Model Based Design to first simulate and then to target a stereo rectification and undistortion algorithm to an embedded FPGA system on chip hardware target. In short, Model Based Design provides a mathematical and visual approach to simulating complex systems as well as the capability to generate executable code from those simulations. This work demonstrates what the future of FPGA-based algorithm development might look like by making use of closed loop design techniques with the aid of novel Model Based Design tool flows. We present the use of hardware-abstracted constructs such as 'virtual cameras' and the capability of pulling images off the target system and feeding them back into the simulation model to help minimize discrepancies between simulation and hardware domains, ultimately reducing the total development time. The developed system is a deterministic real time and low latency implementation as it does not rely on external off chip memory as all necessary buffering is done in the internal FPGA block RAM structures. The design is capable of processing 752×480 images at pixel line rates and is easily expandable to high resolution images. The algorithm being developed is not being touted as novel, rather that it is sufficiently complex in order to demonstrate the power of the Model Based Design tool flows.

INDEX TERMS Stereo vision, FPGA, lens undistortion, rectification, model based design.

NOMENCLATURE

CAM	Camera.
I_{in}	Input Image.
M_{rec}	Rectified Image Mapping.
M_{comp}	Compensated Image Mapping.
I_{out}	Output Image.
Int	Intrinsic Matrix.
Ext	Extrinsic Matrix.
H	Homography Matrix.
H^{-1}	Inverse Homography Matrix.
f_x	X Axis Focal Length.
f_y	Y Axis Focal Length.
c_x	X Coordinate of the Optical Center.
c_y	Y Coordinate of the Optical Center.
$skew$	Skew Factor of a Camera.
$k1$	Radial Distortion Coefficient 1.
$k2$	Radial Distortion Coefficient 2.
$p1$	Tangential Distortion Coefficient 1.
$p2$	Tangential Distortion Coefficient 2.
SoC	System on Chip.
$FPGA$	Field Programmable Gate Array.
$BRAM$	Block Random Access Memory.

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Wei.

I. INTRODUCTION

In recent years there have been great advances in the fields of artificial intelligence and autonomous systems. One of the driving technologies behind these advances is the field of

computer vision. In short, computer vision gives an electronic system the capability to 'see' the world just as a human does. This paper focuses on a subset of computer vision in which two or more cameras are used. The benefits of using a multi-camera system is that important information about the environment such as depth can be determined without the need for any active sensing elements which LiDAR and structured light approaches require. There are however some technical hurdles such as camera calibration that must be implemented in order to create a useful stereo camera system. Calibration typically encompasses determining camera parameters which are used for image rectification and undistortion which are necessary steps in the image processing pipeline. This paper demonstrates a rapid way to simulate, generate, target, and verify a FPGA-based stereo video preprocessing system by way of Model Based Design with the MathWorks HDL Coder™ tool flow.

A. FPGA MODEL BASED DESIGN

The main thrust of Model Based Design is that all aspects of a design are centered around a common model that is shared across a development team. System requirements can be captured and a high level systems model can be simulated with tools like MATLAB® and Simulink®. Once the simulations match the requirements, the model can be compiled into either C code, HDL, or CUDA code via Simulink® Coder™, HDL Coder™, or GPU Coder™ products. The compiled HDL can then be simulated against a plant model in a closed loop fashion with tools such as HDL Verifier™. This common model approach shortens development time and eliminates bugs as production code can be simulated against hardware models before the hardware is available. In this work, a model of a stereo rectification and undistortion system is first simulated and then converted to HDL via the HDL Coder tool. The HDL is compiled to a bit file which is then targeted to the FPGA-based camera system. A 'virtual camera' capability was developed and utilized which helped to verify algorithm functionality on the embedded system. Ultimately the raw camera feeds were fed directly into the auto-generated FPGA stereo rectification and undistortion algorithm to demonstrate the complete system functionality.

As mentioned above, the focus of this article is the closed loop capability of Model Based Design to simulate, generate, target, and verify FPGA-based algorithms. The stereo rectification and undistortion algorithm was chosen as a candidate algorithm to demonstrate the power of Model Based Design for several reasons. Stereo rectification and undistortion must be processed in real time, it is a very important step in the depth-from-stereo image processing pipeline, and it is quite advanced as it makes use of block RAM elements as well as high precision and high dynamic range data paths. The theory, background, and related work pertaining to stereo rectification and undistortion is discussed below to give one context, however all main contributions of this article are pertaining to Model Based Design tool flows and techniques and are listed below.

- Creation of a closed loop Model Based Design development process by reading frame buffers directly off an FPGA-based camera and into a Simulink model via Frame to Pixel blocks.
- Development of a 'virtual camera' capability which allows one to inject synthetic camera frames into the FPGA image processing pipeline. These frames are the same frames that are fed through the Simulink simulation model.
- Demonstrating the above two contributions by simulating, generating, targeting, and verifying a stereo rectification and undistortion algorithm with MathWorks tools and an FPGA-based stereo camera system.

B. STEREO RECTIFICATION AND UNDISTORTION

The high level stereo rectification and undistortion algorithm employed in this article can be seen in Fig. 1 which shows the inputs, data flow, and outputs for a single camera feed. The homography matrix and distortion coefficients are input parameters which are determined during the calibration process. The processing for the left and right cameras is exactly the same and only differs based upon these input parameters. The final interpolation step is required since the backwards pixel mapping from the output image back to the input image does not occur neatly on pixel boundaries. The entire processing pipeline is marshalled by the input image control lines which are the same for each camera since the cameras are synchronized in hardware. Further details of the image processing pipeline are expanded upon in the following sections.

II. RELATED WORK

While computer vision is relatively new compared to the broader field of optics, it builds upon the same mathematical principles and uses software or hardware algorithms to achieve one's desired result. The process of undistortion and rectification is not new in the computer vision field, however the use of FPGAs in this process (especially in video) is relatively new with research being emphasized in the past two decades [1], [2]. FPGAs are appealing largely due to their flexibility, energy efficiency, low latency, and high throughput. The latter is especially poignant as many researchers highlight the main goal of their research being real time rectified and undistorted video [3], [4]. Most of the research has emphasized implementing various algorithms onto FPGAs [5], [6]. Some researchers focus on security applications where the intent is to stitch multiple camera feeds together [7] while a great deal of research has been focused on the field of stereo imaging [4], [8]. Some have proposed implementing neural networks onto FPGAs that perform computer vision tasks, such as Loni et al., but their work is only in the initial stages and is often limited by the size of the FPGA and requires external memory to operate [9].

Two competing approaches have been proposed when implementing rectification and undistortion algorithms. Some implement the necessary mathematical equations with

arithmetic circuits [2], [4], [10], while others advocate for the use of lookup tables that perform similar functions to the equations [11], [12].

The lookup table approach usually involves storing information either on chip or in separate memory which does tend to make designs doable on smaller FPGAs, however it does complicate the overall process and requires one to update these memory locations. Another downside of needing to use reverse transformation lookup tables is pointed out by [13] which states that for a 1024 × 1024 image, it is necessary to store 8 MB of lookup table data. Some advances such as lookup table compression and subsampling have been explored by Junger et al. [1] and [14] respectively however the tables still grow proportionally as image sizes increase.

FPGAs have increased in gate count while becoming cheaper and thus our approach relies on this fact to simply compute the necessary mathematical equations on the fly within the programmable fabric. Not relying on external memory not only makes the design easier, it also allows for a much faster and highly deterministic implementation.

As shown above, multiple researchers have been able to implement rectification algorithms on FPGAs for several years now [2]. The accomplishment of applying these algorithms to hardware is highlighted by the historic difficulty in working with FPGAs and HDL. Early researchers simply hand-coded the necessary HDL which is often time consuming and tedious to debug [15]. Other researchers have proposed various solutions to speed up the development process to still benefit from the speed and flexibility of FPGAs. Several researchers have proposed using C-like languages such as Michalik et al. and Qamar et al., that can be compiled to HDL [2], [16]. Although C-like languages have been proven to speed up the initial development cycle, the approach only improves the coding process and not the simulation and verification process. Conversely, languages like Python and libraries such as OpenCV may allow for algorithm simulation, but converting the simulations into HDL is cumbersome at best.

The concept of auto-generating HDL from various high level programs in nothing new as tools like Xilinx System Generator and the MathWorks HDL Coder have been around since 2012. In the mid 2010s Qamar et al. showcased the usability of high level synthesis and its competitive abilities against traditional manually coded designs [16], [17]. An integrated tool flow for simulating and generating HDL for computer vision algorithms was created in 2019 with the advent of the Vision HDL Toolbox™ which allows one to convert images composed of rows and columns into pixel streams that can be processed at the pixel line rates. Bilal et al. demonstrated the use of this new toolbox by implementing a perspective transformation for bird's eye view generation [18].

In this paper we demonstrate the use of a hybrid fixed and floating point algorithm that employs the mathematical approach of determining the rectification and undistortion inverse mapping as opposed to the lookup method. The entire

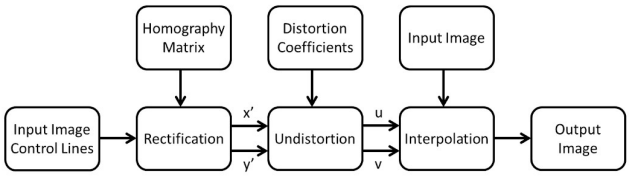

FIGURE 1. Image Rectification and Undistortion High Level Algorithmic Block Diagram.

algorithm exists within the Simulink development environment and the Vision HDL toolbox is used to aid in the simulation by allowing the conversion of images back and forth between logical arrays and pixel streams. HDL Coder is used to convert the algorithm to HDL which is then targeted to an embedded FPGA SoC. The targeted design is then verified via closed loop techniques such as using virtual cameras and pulling frame buffers from the target system. In this article these crucial steps of simulation, HDL generation, hardware targeting, and algorithm verification are incorporated into a single tool ecosystem via a closed loop Model Based Design approach.

III. THEORY

The proposed system relies upon established theories and mathematics of optics to implement a computer vision system. While it is common to refer to the process of camera calibration as encompassing the processes of rectification and undistortion, we limit the use of the term to describe the process of capturing images which are used to calculate the calibration parameters which include intrinsic and extrinsic camera properties. Additionally we rely upon a linear camera model which simplifies calculations by making certain assumptions about how points are translated between 3D space to a 2D image plane.

As [19] discusses, many physical aspects of the camera system directly influence the scene captured and must either be measured or calculated. These properties can include aspects such as focal length, optical center, and distortion (both radial and tangential). While a pinhole model does not perfectly encapsulate all parameters in a camera system, it is generally assumed to be accurate enough [20] as the math is only modified slightly by the inclusion of distortion values which are included in already present translation matrices.

After calibration parameters have been determined, the proposed system is able to perform image rectification and undistortion, which is the process by which new images are created that overcome the relative offsets of the two image sensors as well as the radial and tangential effects. This can be seen in Fig. 2 where two input images (left and center) are converted into rectified and undistorted images (right). The following sections describe the mathematics of the general process described above.

A. CAMERA MODEL

An abbreviated visual representation of a pinhole camera model can be seen in Fig. 3. In it, one sees that point P

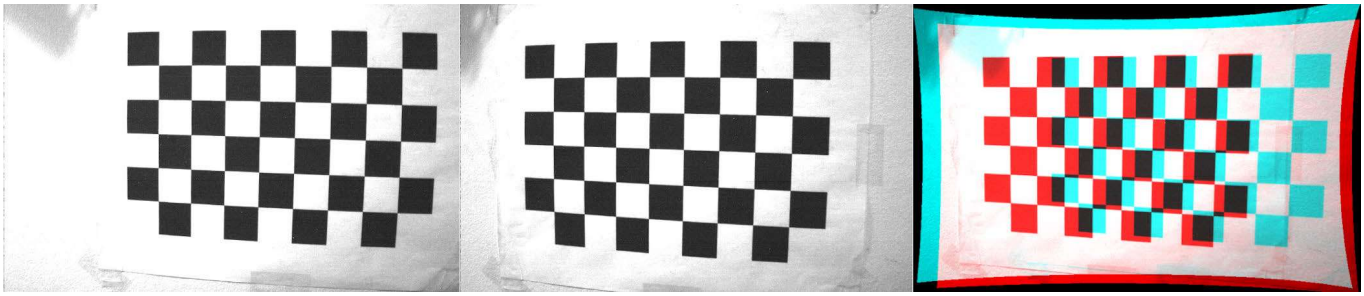

FIGURE 2. Left and Right Simulation Input Images and Associated Simulated Output Anaglyph.

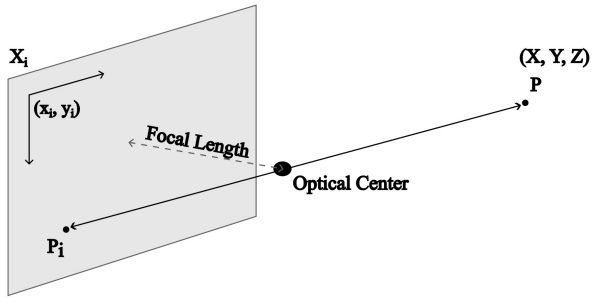

FIGURE 3. Pinhole Camera Model Showing Projection of a Point in 3D World Coordinates into Camera Pixel Coordinates.

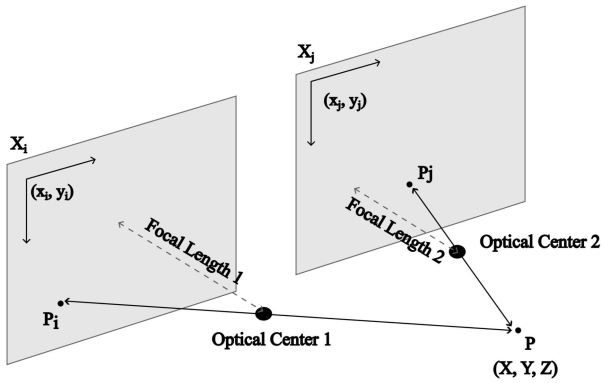

FIGURE 4. Pinhole Camera Model Showing Projection of a Point in 3D World Coordinates into Stereo Camera Pixel Coordinates.

is represented in 3D space in the world coordinate system of (X, Y, Z). Point P is represented on the image plane, X_i as point P_i with coordinates (x, y). All light which passes through this system passes through the optical center.

The proposed system has a stereo camera pair, which can be modeled using the same components as seen in Fig. 4. Point P is still represented in world coordinates of (X, Y, Z) but is now represented as 2 separate points P_i and P_j on image planes X_i and X_j with coordinates of (x_i, y_i) and (x_j, y_j) respectively. Each camera has its own focal length and optical center which are used to produce intrinsic and extrinsic matrices.

The MATLAB stereo camera calibration tool was used to calculate the intrinsic and extrinsic properties for both cameras. The focal length and optical center values make up the intrinsic matrix, and the extrinsic matrix consists of the rotation and translation values that map the point P into camera space. Each camera has both an intrinsic and extrinsic matrix denoted as *Int* (As denoted in (1)) and *Ext* (As denoted in (2)) respectively.

$$\begin{bmatrix} f_x & skew & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = IntrinsicMatrix \tag{1}$$

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \end{bmatrix} = ExtrinsicMatrix \tag{2}$$

B. INVERSE GEOMETRIC TRANSFORM

In a single camera system, the *CAM* produces an I_{in} which is used as an input into the rectification and undistortion process. With the stereo system, there is no way to ensure that the two input image planes are the same. To overcome any discrepancies in placement of the two image planes, a mathematical third virtual camera plane, CAM_V is assumed in which both input images are projected onto resulting in a rectified image mapping, M_{rec} per camera. Unlike [20], both the left and right images are rotated by half in order to reduce the overall error. Since these rotations are done off-line and are boiled down into homography matrices there is no additional latency accrued with this technique. This geometric transformation is accomplished by using the inverse homography matrix of each camera. The equation that describes the transformation from the rectified coordinate system to the unrectified coordinate system can be seen in (3).

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x_{rec} \\ y_{rec} \\ 1 \end{bmatrix} * H_{3\times3}^{-1} \tag{3}$$

It should be noted that (x_{rec}, y_{rec}) represents an integer pixel location that correspond to pixel locations on the right side of Fig. 5 whereas (X, Y, Z) represents homogeneous coordinates corresponding to the input *CAM* feed. Note that the left side pixel Cartesian locations in Fig. 5 are denoted as

(x', y') and are derived by (4), (5).

$$x' = \frac{X}{Z} \tag{4}$$

$$y' = \frac{Y}{Z} \tag{5}$$

The (x', y') coordinates are shown in Fig. 1 as the output of the rectification block. It is important to understand that the (x', y') coordinates are the inverse mapping from the soon-to-be-generated rectified image coordinates (x_{rec}, y_{rec}) on the virtual camera plane CAM_V mentioned above to the original left and right images. The (x', y') pixel coordinates are typically sub-pixel values however, and therefore their intensity values must be arrived at via interpolation. For example, the value of the (x_{rec}, y_{rec}) coordinates for the central image pixel would be [376,240], and for our specific camera system it was mapped to (x', y') values of [371.51, 231.44]. This point to point backwards mapping is done for each and every pixel in the rectified image which is then further augmented by the undistortion mapping.

C. UNDISTORTION

Buffering of the input images into memory is commenced at the start of the inverse geometric transform process. When enough lines have been buffered, based upon the maximum distortion and misalignment values, the undistortion portion of the algorithm is performed. Pre-processing for the undistortion process entails normalizing the coordinates (x', y') by translating them to the optical center (c_x, c_y) and dividing them by the focal lengths of the camera (f_x, f_y) (6) [13].

$$(x_n, y_n) = \left(\frac{x' - c_x}{f_x}, \frac{y' - c_y}{f_y}\right) \tag{6}$$

The two main sources of distortion that occur in our system due to lenses come in the form of radial and tangential distortion. Radial distortion is due to light bending differently near the edges of the lens compared to the center, and tangential distortion is due to the lens and sensor not being perfectly parallel [21]. As such, each undistorted image plane with coordinates (u, v) has radial and tangential components. While this process will improve all images taken with a lens component it is most noticeable in systems utilizing wide angle lenses.

The rectified mapping M_{rec}, with normalized coordinates of (x_n, y_n) are put through the undistortion block which walks through the following equations. While dense in nomenclature, the equations use well established theory from the imaging field. First the value of r^2 is calculated in (7),

$$r^2 = x_n^2 + y_n^2 \tag{7}$$

which is then utilized in conjunction with radial and tangential coefficients to calculate the level of distortion on each point respectively in (8), (9), (10), (11).

$$u_{radial} = x_n(1 + k1r^2 + k2r^4) \tag{8}$$

$$u_{tangential} = 2p1x_ny_n + p2(r^2 + 2x_n^2) \tag{9}$$

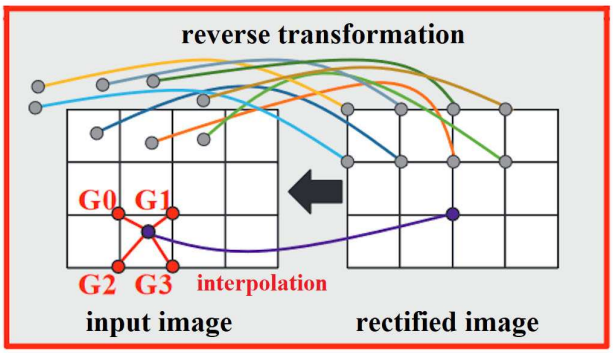

FIGURE 5. Reverse Transformation and Interpolation Process [Nomenclature slightly modified] [1].

$$v_{radial} = y_n(1 + k1r^2 + k2r^4) \tag{10}$$

$$v_{tangential} = 2p2x_ny_n + p1(r^2 + 2y_n^2) \tag{11}$$

Finally the coordinates (u, v) are calculated by combining the tangential and radial portions of each in (12) and (13).

$$u = u_{radial} + u_{tangential} \tag{12}$$

$$v = v_{radial} + v_{tangential} \tag{13}$$

This process yields mappings M_{comp} that will produce an image with coordinates of (u, v) that have been compensated for lens distortion. To continue with the example from above the (x_{rec}, y_{rec}) coordinates for the central image pixel of [376,240] which are mapped to (x', y') values of [371.51, 231.44] are ultimately mapped to (u, v) coordinates of [349.42, 231.51]. The sub-pixel (u, v) values are then fed into the interpolation stage to pull out the proper pixel intensity values from the original input image. Identical mathematical steps are taken with the left camera and its associated variables.

D. INTERPOLATION

While the M_{comp} mapping is no longer distorted and is in the correct coordinate system, those coordinates may not line up exactly with pixels in the input image. Therefore the mapping must be split up to the surrounding 4 pixels via a process known as bilinear interpolation. The interpolation is shown in (14-17) and shows how a point (u, v) is represented by the surrounding pixels G_0, G_1, G_2, G_3 utilizing *deltaU* and *deltaV* which describe the location of the sub-pixel in relation to the nearest pixels.

$$compPixel = G_0(1 - deltaU)(1 - deltaV) \tag{14}$$

$$+G_1(1 - deltaU)(1 - deltaV) \tag{15}$$

$$+G_2(1 - deltaU)(1 - deltaV) \tag{16}$$

$$+G_3(1 - deltaU)(1 - deltaV) \tag{17}$$

Fig. 5 created by Junger et al. [1] accurately shows the process of bilinear interpolation where the point is in the middle of the G pixels and must be spread among the four pixels to be represented accurately. This final process produces the image I_{out} which has successfully been undistorted and rectified.

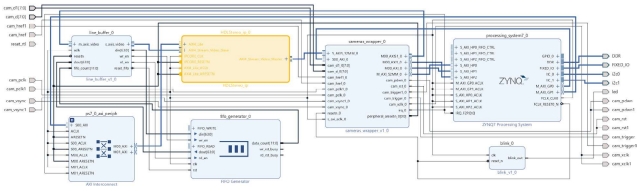

FIGURE 6. Xilinx Vivado Block Diagram with Rectification and Undistortion IP Core Highlighted in Orange.

IV. SOFTWARE DEVELOPMENT TOOL ECOSYSTEM

A. VIVADO

The Xilinx Vivado 2019.1 tool was used for HDL development for this project. The static parts of the design include the camera pre-processing stages such a demosaicing as well as various line buffers. The Vivado HDL block diagram for this project can be seen in Fig. 6. The dynamic portion of the design which is auto-generated is shown in orange. One can think of the entire project as containing peripheral wrapper code as well as main algorithmic core code.

B. MATLAB AND SIMULINK

All simulation and HDL code generation was performed with the MathWorks tools namely MATLAB and Simulink. Additional toolboxes were required such as the Fixed-point designer as well as the Vision HDL Toolbox and HDL Coder.

C. VIVADO SIMULINK TEMPLATE

A Zynq 7020 was used for this research which requires the Vivado toolchain for bit file compilation. A reference design was created in Vivado that was exported into a Simulink HDL reference design. In this manner one can simply select the Vivado reference design under the IP core generation tab and Simulink will take care of generating the IP core as well as the entire bit file compilation. This is highly beneficial as different image processing cores can be simulated and auto-generated without the developer ever opening the Vivado tool.

D. ORACLE VIRTUALBOX

Oracle VirtualBox was used in order to generate the Petalinux image for the Zynq SoC. A stand alone Linux PC could have been used for this task, but it was much easier to use a Virtual Box which allows for easy file copying between the host PC and the virtual Linux system. A Petalinux Board Support Package [BSP] was created and then a PYNQ image was created by Bootstrapping Ubuntu 18.04. The resultant PYNQ Linux image was then deployed to the Zynq SoC where user space programs were written to marshal the FPGA-based vision processor.

V. HARDWARE TARGET

A FPGA-based stereo camera system was developed for this work that was built around the Xilinx Zynq 7020 SoC. The Zynq 7020 has a dual ARM Cortex-A9 that runs at 1 GHz as well as a programmable FPGA fabric. The system is composed of two hardware synced MT9V034 global shutter cameras that are capable of operating at 60 FPS with a resolution of 752 × 480 pixels. The cameras feed directly into the FPGA fabric where the images are processed in real time and are then fed up to the Linux processor by Xilinx VDMA engines. Once the images are read in via the Linux processor, they are compressed and output via WiFi to MATLAB or a web page where they can be viewed remotely in a web browser on the host PC. The high level system topology can be seen in Fig. 7 below. Note that Fig. 7 only shows a single camera input as it is abstracted to better show the algorithm development process which is discussed in Section VI. The detailed image processing flow is discussed below.

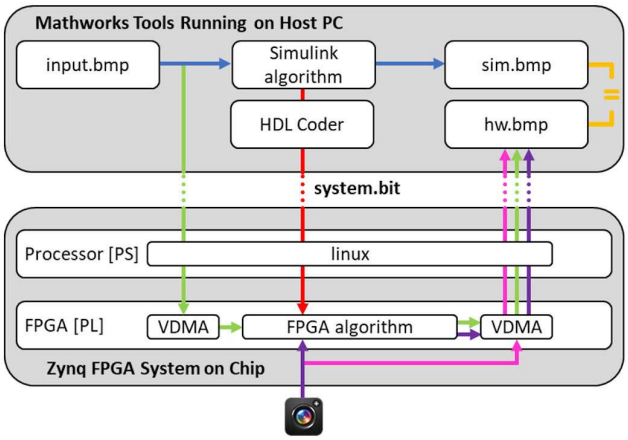

FIGURE 7. Closed Loop Model Based Design Algorithm Simulation, Generation, Targeting, and Verification High Level Flow Diagram. Dashed Lines Represent WiFi.

A. IMAGE PROCESSING FLOW

As mentioned above, the input images are fed into the FPGA fabric where some baseline image processing such as demosaicing is performed. The left and right images are then combined into a single 64 bit AXI stream that is composed of the red, green, and blue [RGB] channels as well as the generated gray channel for each camera. This 64 bit raw AXI video stream is sent up to the processor as shown in Fig. 8 but it is also split off into a second channel that is sent into a multiplexer. The purpose of the multiplexer is to select either input images from the cameras or images sent down from the processor to be sent to the image processing core. This is a very important feature as when developing algorithms it is essential to be able to fix the inputs in order to debug the processing stage. In this manner, one can verify their processing core with static images from the processor via a virtual camera, and then switch the multiplexer to process images from the live camera feeds.

The Image Processing Core is auto-coded with the HDL Coder tool and contains an HDL version of a user-defined Simulink model. Both the input and processed image feeds are then sent up to the processors via an AXI VDMA Xilinx IP block. Linux drivers were written in the C programming

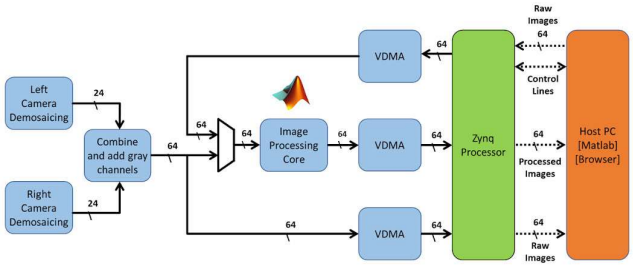

FIGURE 8. Detailed Closed Loop Data Flow Diagram. Blue: Zynq PL, Green: Zynq PS, Orange: Host PC, Dashed Lines: WiFi.

language that read in both the input and processed image feeds. The image feeds are then sent to the host PC via a WiFi connection as shown in Fig. 8.

VI. METHODOLOGY

There are six distinct Model Based Design flows that were used for the design and development of the candidate stereo rectification and undistortion algorithm. Each flow is shown in Fig. 7 and is depicted with various colored lines. To simplify comprehension, the design flows are discussed at first via a simplified example below followed by their actual implementation later on in the section. Assuming one wanted to perform a simple Sobel filter on an image, they would follow the below flows.

1) Grab Raw Frames from Camera.
   - The developer would want an example image to use as an input to their simulation. The flow in pink shows how a raw image can be pulled off the camera and saved as hw.bmp. This image can be renamed to input.bmp which is used in the next flow.
2) Develop Algorithm Based on Raw Frames.
   - The flow in blue shows how a Simulink model is created that uses input.bmp as the input image and outputs the Sobel filtered image as sim.bmp. Once the algorithm is correct it is auto-generated into HDL in the next flow.
3) Auto-Generate HDL and system.bit files.
   - HDL Coder is used to auto-generate HDL from the Simulink Sobel filter block and the red flow shows how the system.bit file is copied to the Zynq SoC.
4) Send Virtual Camera Image into FPGA Algorithm.
   - Now that the auto-generated algorithm is loaded into the FPGA, the input.bmp file is sent down to the hardware and into the FPGA-based Sobel filter. The output of the FPGA Sobel filter is captured as hw.bmp. This flow is depicted via the green lines.
5) Verify FPGA Results with Simulation Results.
   - The output of the FPGA-based Sobel filter [hw.bmp] is compared to the output of the Simulink simulation [sim.bmp] and if the images match then the FPGA-based algorithm has successfully been verified. This flow is depicted via the yellow lines.
6) Feed Camera Image into FPGA Algorithm.
   - After verifying the algorithm with the virtual camera, the live camera feed is fed into the FPGA-based Sobel filter algorithm as is shown via the purple lines.

The closed loop nature of the above Model Based Design flow can quickly find bugs which can then easily be isolated between requirements issues, hardware issues, algorithm issues, etc. The rest of the section walks through the above steps but with regard to the stereo rectification and undistortion routine.

As mentioned in the theory section each camera has several properties such as optical center and focal lengths that are unique to that camera. These properties can vary a great deal even in the presence of tightly held production tolerances. To achieve the highest level of performance for stereo imaging, the cameras must be calibrated both individually and in relation to each other. There are several steps that must be taken in order to successfully calibrate a stereo camera pair. This section walks one through each step in a chronological fashion which ultimately results in a fully calibrated stereo camera system.

A. DETERMINE CAMERA INTRINSICS AND EXTRINSICS

The first task is to determine the properties of the left and right cameras namely their intrinsic and extrinsic matrices. The intrinsic parameters are internal to each camera and are shown in (1) above. The extrinsics of a camera describe the absolute pose of the camera relative to a fixed object such as a planar surface in this case.

To determine the intrinsics and extrinsics of the stereo camera system, several images of a calibration checkerboard were taken. These images were taken by interacting with the live video feed web page that is served up by the Linux processors of the hardware target. These images were taken via the processing flow depicted in pink in Fig. 7 or equivalently via the Raw Images line depicted in Fig. 8.

Once approximately 10 pairs of left and right calibration images are captured, they are fed into the MATLAB Stereo Camera Calibrator application. This application analyzes the group of left and right image pairs and automatically determines all of the pertinent calibration parameters. Fig. 9 shows a sample output of the Stereo Camera Calibrator which shows the location of the calibration image relative to the camera system.

B. SIMULATE RECTIFICATION AND UNDISTORTION IN SIMULINK

The output of the Stereo Camera Calibrator application is a MATLAB structure called stereoParams that contains both the intrinsics and extrinsics of the stereo camera system. This structure is then processed to arrive at the left and right camera inverse homography matrices. These matrices

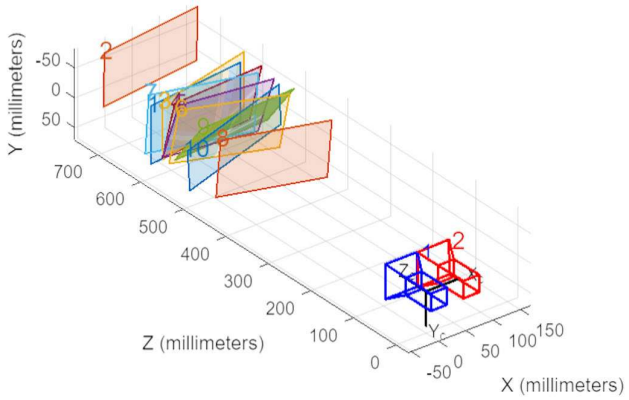

FIGURE 9. Calibration Image Locations as Determined by the MATLAB Stereo Camera Calibrator Tool.

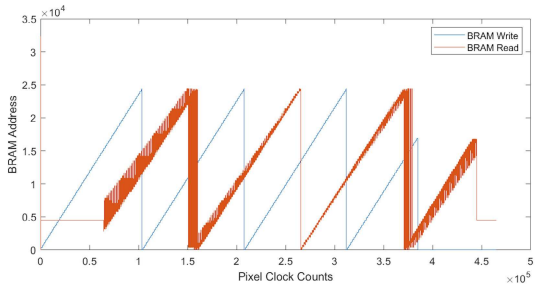

FIGURE 10. Simulink Simulation of Block RAM Read and Write Addresses During the Rectification and Undistortion Process.

along with the radial and tangential distortion parameters are fed into the Simulink model as parameters as is shown in Fig.1. After the rectification and undistortion operations are performed, the last step is the interpolation stage which effectively determines which address of the BRAM to read out.

Fig. 10 shows the read and write addresses of one of the four interpolation BRAMs. For this example there were 80 rows buffered which is the reason for the phase delay between the blue and orange lines. The input image is first written to the BRAM and then after the correct number of rows are buffered, the rectified output image starts to read from the BRAM according to the address determined by the interpolation routine.

The entire Simulink model used in this paper can be seen in Fig. 11 below. The starting point of the model was a Simulink reference design which was then modified for our purposes. The model effectively loads in left and right image files and then converts the image arrays to pixel streams via the Vision HDL Toolbox. These pixel streams are concatenated into a single 64 bit pixel stream and are fed into the middle block which is converted into HDL. Only a single control line is used as both the left and right cameras are hardware synchronized. The output of the middle block is a 64 bit pixel stream that is then parsed to pull out the left and right processed pixel streams. Finally the pixel streams are converted back into image arrays and are then sent to Vision HDL Toolbox display blocks.

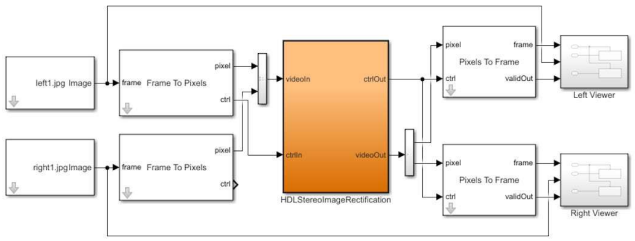

FIGURE 11. Top Level Simulink Simulation Model.

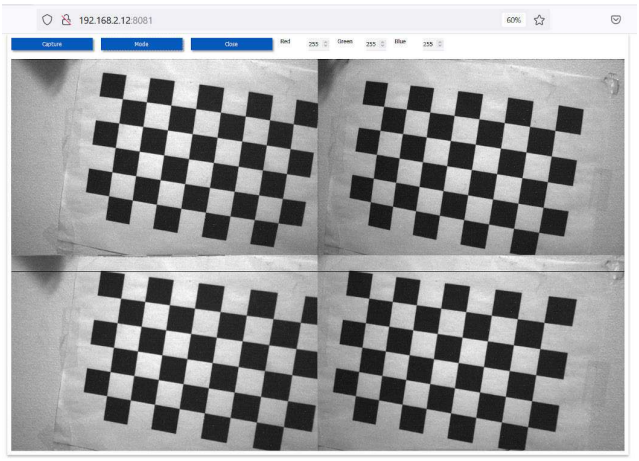

FIGURE 12. Live Left and Right Raw as well as Left and Right Rectified and Undistorted Video Feeds Streaming from the Zynq SoC to a Web Browser.

The simulation process described above is depicted in a simplified manner in blue in Fig. 7. Input.bmp would be replaced with raw images collected from the cameras for calibration and sim.bmp would be the rectified and undistorted images produced by the simulation.

C. GENERATE HDL VIA HDL CODER

Once the simulation model was working correctly, the middle block highlighted in orange was converted into HDL via the HDL Coder tool. Note that the 64 bit input and output of the middle block matches the bus widths of Fig. 8. Effectively the middle block is converted into HDL and is plugged into the image processing pipeline as the Image Processing Core. This process is completely automated within the Simulink environment with the use of the Vivado template that was discussed in Section IV above. This is very convenient as Simulink can generate the FPGA bit file without opening up the Vivado tool.

D. FLASH HARDWARE AND STREAM VIDEO

After bit file generation is complete, the file is copied to the */boot* directory of the Zynq SoC via a samba share and the system is rebooted. This flow of generating HDL, compiling a bit file, and targeting the system.bit file to the FPGA is depicted in red in Fig. 7. Upon reboot the FPGA is updated and flashed with the newly auto-generated bit file. A video streaming server is then started and the user can log onto a

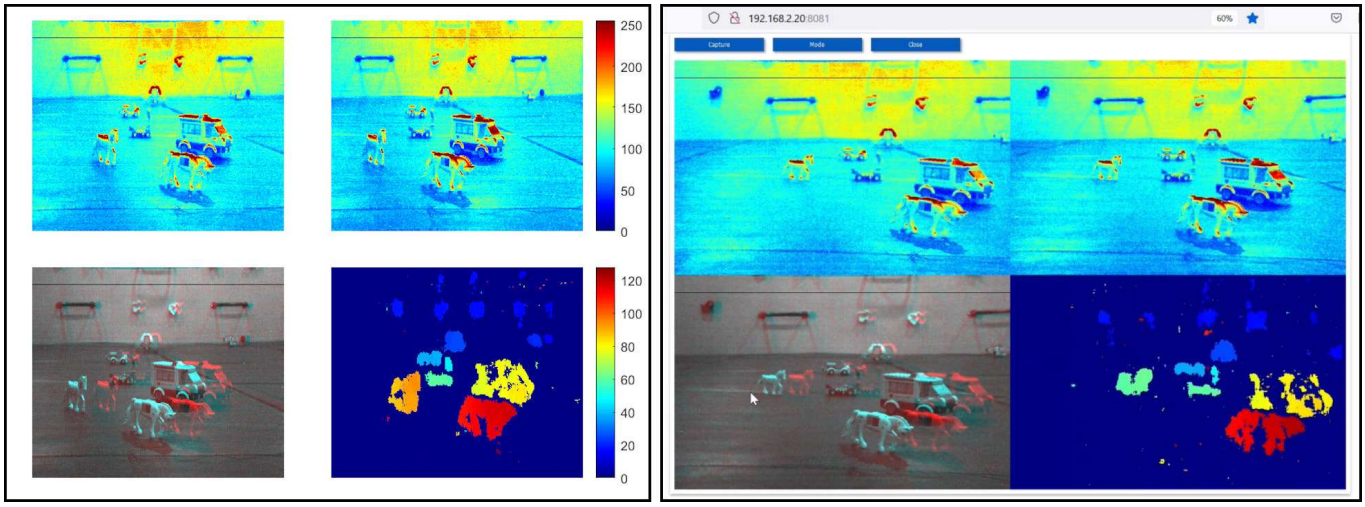

FIGURE 13. Stereo Rectified and Undistorted Images [top], and Stereo Anaglyph and Disparity Maps [bottom] Generated via MATLAB [left] and via OpenCV [right].

web page via their host PC in order to see both the input and processed video streams.

E. VERIFY ALGORITHM WITH VIRTUAL CAMERAS

Once the bit file has been flashed onto the FPGA, the next step is to verify that the algorithm works when it is fed with the very same inputs as the simulation model. This process is depicted via the green lines in Fig. 7. As stated above, Fig. 7 is for conceptual purposes and a raw stereo image pair is sent down rather than the input.bmp image. The simulation images are sent through the FPGA-based auto-generated algorithm and the outputs [depicted as hw.bmp] are collected back in the simulation domain. In reality the output images would be composed of the left and right rectified and undistorted stereo pair. These images are then compared against the simulated outputs as depicted via the yellow line in Fig. 7. This virtual camera capability is very beneficial as developers with different camera systems and optical properties can both debug and tweak the algorithm. This is due to the fact that the entire photon to pixel pathway is short-circuited and common testing and simulation image pairs can be used across development teams. Another advantage of the virtual camera is that the same raw image pair can be used over and over again, whereas when working with live camera feeds, the input images are different every frame which makes it difficult to isolate variables for debugging purposes. Once the FPGA-based algorithm has been verified with the virtual cameras the live camera feeds are used as depicted via the purple lines in Fig. 7. The results from the live camera feeds are discussed in the next section.

VII. RESULTS

The resultant video streams can be seen in Fig. 12. The top images correspond to the left and right input images and the bottom images correspond to the left and right processed images. A debugging black line was placed at row 40 of both processed images to clearly show that the images have been successfully rectified. One can also see by holding up a straight edge to the image boundaries that the curvature present in the upper images is completely removed in the lower two images. This demonstrates that the radial distortion has been corrected for. Users can click "Capture" at the top of the web browser to capture a complete set of both input and processed image pairs. This is very useful as discussed above as the input images can then be fed back into the Simulink model to verify that the simulation outputs match the processed images captured off of hardware.

TABLE 1. Comparative Zynq 7020 Resource Utilization Between Kaputa and Junger Algorithms. Junger's values are twice what was published as this table shows the sizing for both the left and right camera IP cores.

Resource	Util. (Kaputa) %	Util. (Junger)%
LUT	26	6
LUTRAM	10	2
FF	19	4
BRAM	46	24
DSP	78	14

The overall size of the stereo rectification and undistortion processing core is shown in Table. 1. As mentioned above, as image size grows in the future, the number of DSP blocks will remain constant as the image is processed in a streaming fashion. The BRAM utilization will rise but only in proportion to the increased row width. This is due to the fact that a certain number of rows must be buffered in order to perform successful inverse mapping from the rectified camera plane back to the unrectified frame.

While our initial BRAM utilization is higher when compared to Junger et al. [1], this is due to some external design differences such as the number of rows being buffered. The real time mathematical processing in our work is the reason for higher LUT, DSP, and flip flop utilization. Our design is far more scalable since it scales based upon image width

whereas Junger's design relies on external memory to store information related to both image width and height. With a 4K image, both of our BRAM utilizations would be comparable however DDR memory utilization of Junger's design would be significantly larger due to the size of the rectification maps needed. In short, as image size grows our BRAM utilization scales with image width (instead of image width and height) and is more likely able to perform well while continuing to remain on chip.

VIII. LIMITATIONS

At present the system described above is only capable of performing grayscale rectification and undistortion. This is due to the fact that there is not enough BRAM storage to buffer the red, green, and blue pixels on the Zynq 7020 SoC. Additionally, there may be scenarios where the left and right image planes are vertically offset by a large amount which could arise during the manufacturing process. Under this circumstance a greater number of rows would need to be buffered to be able to perform successful inverse mapping. It is quite possible that there would not be enough BRAM resources in the FPGA fabric to accommodate the extra row buffering.

IX. CONCLUSION AND FUTURE WORK

This work demonstrates a Model Based Design flow for the generation of a real time FPGA-based stereo processing core. The bit file was generated without the need to open the Vivado tool ecosystem as all the processing was marshalled by the Simulink code generation engine. The auto-generated core was successfully run on a Zynq 7020 SoC and closed loop techniques and virtual cameras were used to aid in debugging and algorithm verification. This design flow enables engineers to reduce their development times and to better link their final HDL algorithm implementation to the high level algorithm simulation model.

Future work entails potentially combining the demosaicing and the rectification operations into a single process as is demonstrated in [22] and using HDL Coder to auto-generate the associated IP core. With larger FPGA fabrics it may also be beneficial to convert the entire algorithm into floating point logic such that the fixed point datapath does not need to be tweaked for various image sensors.

Given more BRAM resources it would also be nice to create a full RGB rectification and undistortion algorithm. This new RGB implementation would also include grayscale and the Linux user application would be able to read either grayscale or RGB stereo frames from the FPGA. Ultimately an entire stereo depth algorithm would be developed in the FPGA such that the Linux user application could directly read out a depth map without using any CPU cycles for disparity mapping.

This work is already under way as the depth algorithm has been developed in both OpenCV and MATLAB and the depth images can be seen in Fig. 13. A basic block matching technique was used via the MATLAB *disparityBM* function and cooresponding OpenCV *StereoBM* function. The parameters used for block matching were a disparity range from 0 to 128, a block size of 19, and a uniqueness threshold of 15. In the case of the MATLAB generated ensemble, the FPGA pre-processed images were read into MATLAB via WiFi and a MATLAB script performed the block matching. In the OpenCV ensemble, the FPGA pre-processed images were read into the Linux processor and OpenCV performed the block matching in real time. The resulting images were served up via a web page in the same fashion that was discussed above. Future work entails optimizing the block matching algorithm such that the entire depth pipeline will fit into the FPGA fabric.

REFERENCES

[1] C. Junger, A. Hess, M. Rosenberger, and G. Notni, "FPGA-based lens undistortion and image rectification for stereo vision applications," *Proc. SPIE*, vol. 11144, pp. 284–291, Sep. 2019. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11144/1114416/FPGA-based-lens-undistortion-and-image-rectification-for-stereo-vision/10.1117/12.2530692.full

[2] S. Michalik, S. Michalik, J. Naghmouchi, and M. Berekovic, "Real-time smart stereo camera based on FPGA-SoC," in *Proc. IEEE-RAS 17th Int. Conf. Humanoid Robot.*, Nov. 2017, pp. 311–317.

[3] Y. Xu, Q. Zhou, L. Gong, M. Zhu, X. Ding, and R. K. F. Teng, "High-speed simultaneous image distortion correction transformations for a multicamera cylindrical panorama real-time video system using FPGA," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 6, pp. 1061–1069, Jun. 2014.

[4] J. Mun and J. Kim, "Real-time FPGA rectification implementation combined with stereo camera," in *Proc. Int. Symp. Consum. Electron. (ISCE)*, Jun. 2015, pp. 1–2.

[5] K. S. Changan and P. G. Chilveri, "Stereo image feature matching using Harris corner detection algorithm," in *Proc. Int. Conf. Autom. Control Dyn. Optim. Techn.*, Sep. 2016, pp. 691–694.

[6] T. Kenter, H. Schmitz, and C. Plessl, "Kernel-centric acceleration of high accuracy stereo-matching," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Dec. 2014, pp. 1–8.

[7] J. Zhu, Y. Li, Z. Ma, Y. Jiao, and B. Zhang, "Study on multiple image processing hardware system based on DSP," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2014, pp. 1844–1849.

[8] S. Perri, F. Frustaci, F. Spagnolo, and P. Corsonello, "Design of real-time FPGA-based embedded system for stereo vision," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[9] M. Loni, A. Majd, A. Loni, M. Daneshtalab, M. Sjodin, and E. Troubitsyna, "Designing compact convolutional neural network for embedded stereo vision systems," in *Proc. IEEE 12th Int. Symp. Embedded Multicore/Many-Core Syst.-Chip (MCSoC)*, Sep. 2018, pp. 244–251.

[10] J. Hyun and B. Moon, "A simplified rectification method and its hardware architecture for embedded multimedia systems," *Multimedia Tools Appl.*, vol. 76, no. 19, pp. 19761–19779, Oct. 2017.

[11] J.-H. Kim, J.-K. Oh, S.-M. Kang, and J.-D. Cho, "A real-time rectification using an adaptive differential encoding for high-resolution video," in *Proc. 16th Int. Conf. Adv. Commun. Technol.*, Feb. 2014, pp. 666–670.

[12] S. N. Hung, J. Lee, and B.-J. You, "Real-time stereo rectification using compressed look-up table with variable breakpoint indexing," in *Proc. 42nd Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2016, pp. 4814–4819.

[13] E. Staudinger, M. Humenberger, and W. Kubinger, "FPGA-based rectification and lens undistortion for a real-time embedded stereo vision sensor," 2008. [Online]. Available: https://www.researchgate.net/publication/228939361_Fpga-based_rectification_and_lens_undistortion_for_a_real-time_embedded_stereo_vision_sensor

[14] P. Di Febbo, S. Mattoccia, and C. D. Mutto, "Real-time image distortion correction: Analysis and evaluation of FPGA-compatible algorithms," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Nov. 2016, pp. 1–6.

[15] D. W. Yang, L. C. Chu, C. W. Chen, J. Wang, and M. D. Shieh, "Depth-reliability-based stereo-matching algorithm and its VLSI architecture design," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 6, pp. 1038–1050, Jun. 2015.
[16] A. Qamar, F. B. Muslim, and L. Lavagno, "Analysis and implementation of the semi-global matching 3D vision algorithm using code transformations and high-level synthesis," in *Proc. IEEE 81st Veh. Technol. Conf.*, May 2015, pp. 1–5.
[17] F. Qamar, F. B. Muslim, F. Gregoretti, L. Lavagno, and M. T. Lazarescu, "High-level synthesis for semi-global matching: Is the juice worth the squeeze?" *IEEE Access*, vol. 5, pp. 8419–8432, 2017.
[18] M. Bilal, "Resource-efficient FPGA implementation of perspective transformation for bird's eye view generation using high-level synthesis framework," *IET Circuits, Devices Syst.*, vol. 13, no. 6, pp. 756–762, Sep. 2019, doi: 10.1049/IET-CDS.2018.5263.
[19] A. Hernandez, A. Gardel, L. Perez, I. Bravo, R. Mateos, and E. Sanchez, "Real-time image distortion correction using FPGA-based system," in *Proc. 32nd Annu. Conf. IEEE Ind. Electron.*, Nov. 2006, pp. 7–11.
[20] B. Maldeniya, D. Nawarathna, K. Wijayasekara, T. Wijegoonasekara, and R. Rodrigo, "Computationally efficient implementation of video rectification in an FPGA for stereo vision applications," in *Proc. 5th Int. Conf. Inf. Autom. Sustainability*, Dec. 2010, pp. 219–224.
[21] D. C. Brown, "Decentering distortion of lenses," *Photogramm. Eng.*, vol. 32, no. 3, pp. 444–462, 1996.
[22] P. Verma, D. E. Meyer, H. Xu, and F. Kuester, "Splatty—A unified image demosaicing and rectification method," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Waikoloa, HI, USA, Jan. 2021, pp. 786–795. [Online]. Available: https://ieeexplore.ieee.org/document/9423304/

DANIEL S. KAPUTA (Member, IEEE) was born in Troy, IL, USA, in 1980. He received the B.S. degree in computer engineering and the M.S. and Ph.D. degrees in electrical engineering from the State University of New York at Buffalo, in 2002, 2004, and 2007, respectively. He worked in industry for ten years at various mil-aero companies, such as the Lockheed Martin Gravity Systems and Moog Space and Defense Group. He is currently an Assistant Professor with the Department of Computer Engineering Technology, Rochester Institute of Technology. He is also the Director of Ravven Laboratories (www.ravvenlabs.com). His research interests include FPGA-based embedded vision, artificial intelligence, unmanned aerial vehicles, and augmented and virtual reality. He has served on the MathWorks Advisory Board for several years and is heavily involved with the use and development of model based design tools. He is also a member of AIAA.

KRYSTIAN A. DERHAK was born in Salt Lake City, UT, USA, in 1996. He received the B.S. degree in computer engineering technology from the Rochester Institute of Technology (RIT), Rochester, NY, USA, in 2021, where he is currently pursuing the M.S. degree in computer science.

He has worked in several internship positions related to embedded systems as a Production Engineer with Critical Link and an Engineer Design Group Intern with MathWorks. In 2022, while at RIT, he joined Ravven Laboratories to work on embedded computer vision applications.