

# Field-Programmable Gate Array Tesla 360° Video View

Michael Popik

Department of Electrical Engineering and Computer Science  
Massachusetts Institute of Technology  
Cambridge, MA, USA  
mpopik@mit.edu

Feyi Adebekun

Department of Electrical Engineering and Computer Science  
Massachusetts Institute of Technology  
Cambridge, MA, USA  
feadebek@mit.edu

**Abstract**—A system integrating images taken from distributed camera angles into a single, unified panoramic image is presented. It leverages an infrared-controlled turntable to continuously rotate a set number of degrees, stop the rotation, and collect an image. These images are then stitched together using perspective transformation homography methods. A rotation-encoded homography can be applied to each image, and the transformed images stitched together to produce a unified panoramic camera view that is sent to an output monitor. The panoramic view updates once all images are available.

**Index Terms**—Field-programmable gate arrays, image processing, homography, image stitching, infrared communication

## I. INTRODUCTION AND MOTIVATION - MICHAEL

Most modern vehicles contain cameras that constantly take images of the surrounding area, in order to improve driver safety. For this project, we wanted to implement a similar system—one capable of taking images from different angles to give the user a unified view—but instead of being implemented using existing expensive hardware, we wanted this to be portable and inexpensive. Unified views can come in a variety of forms: panorama, birds' eye, and mesh, so we sought to implement each of these views in order of increasing difficulty, going from a working panorama to eventually a working three-dimensional mesh view. In prior exercises involving cameras, our data was streamed to either a block RAM (BRAM) or distributed RAM (DRAM), but no intermediate transformations were performed, nor was the data stored for an extended period of time. In our case, incoming pixels must be first transformed, then stored non-consecutively in a DRAM according to their transformed coordinates, and then read out after many seconds, i.e. after all the images that were to be integrated into the view were taken. So, this project presents a divergence from known methods and makes interacting with the previously written DRAM modules much more difficult.

For our technical requirements, as previously mentioned, we envisioned three methods of image transformations and stitching to create unified views: panorama, birds-eye, and three-dimensional mesh. To allow for images to be taken at differing angles, we wanted to mount our FPGA and camera

module upon a turntable that could be controlled by infrared signals sent by a remote. Instead of simply using the remote to control the turntable, we wanted the FPGA to emulate these signals, allowing for a unified state machine where the FPGA could signal that a transformed image had been captured, initiate a rotation, stop the turntable, and repeat this process according to a given set of commands. Our current system achieves this, both rotating and stopping the FPGA. It also processes homographies on the input image to create a unified view, although this could still benefit from blending.

## II. TOP LEVEL DESIGN - FEYI AND MICHAEL

A typical operation cycle of our machine looks like the following:

Rotate → Capture → Rotate → Capture → ... → Rotate → Blend.

During this sequence, the turntable control, rotation, and data streaming all need to be coordinated. The turntable control needs to transmit signals in the Hz range, but the clocks we have used on the FPGA work in the MHz range. Its operation is detailed in the state machine below:

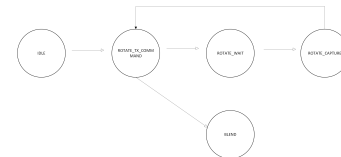


Fig. 1. Top Level FSM

Let us illustrate an example. Suppose we want to take seven images to integrate into a panoramic view. To simplify things, we only allow for odd numbers of images to be taken so we can have a central image and then an equal number of images at an angle and its inverse. The turntable rotates roughly twenty-five degrees per second, and we sent a command to stop once a second. Image 1 is taken at 0°. Then, Image 2 is taken at 25°, Image 3 is taken at 50°, and Image 4 is taken at 75°. After this, we have to rotate -100° degrees to take Images 5, 6, and 7 at -25°, -50°, and -75°, respectively. This is how the turntable is directed to take all the images.

### III. TURNTABLE INFRARED CONTROL - MICHAEL

To collect and integrate images from different angles into the panorama, the camera module is mounted on a rotating turntable. An infrared (IR) remote controls the turntable, issuing the following six commands: start rotation, stop rotation, rotate counterclockwise, rotate clockwise, speed up, and slow down. Automating the turntable removes the need for the user to point and issue commands using the provided remote. Instead, the system calculates the current angle using the initial angle and rotational speed. It stops at pre-determined angles, an image is captured, and the process continues according to the given specification. For instance, the user might want to view a panorama using four images taken over 180°. So, the turntable should, for instance, rotate clockwise, take two images separated by 45°, rotate counterclockwise by 135°, then take another two images separated by 45°. Using this sequence, the camera provides the necessary image inputs to produce the panorama. In this manner, the user is divorced from the proper operation. To satisfy this, a module must be constructed to emulate commands that would otherwise be sent by the provided remote. This process is described in the following sections.

#### A. System Hardware and Command Logging

To receive commands, we need an IR receiver. Inversely, to send commands, we need an IR transmitter. Our system will be mounted on a turntable whose photodiode for detecting incoming signals is located on its side. Infrared is directional. Everyone who has ever been frustrated trying to turn on their television while pointing their remote in the slightly wrong direction will understand this. So, we are relying on IR reflection from nearby surfaces to direct our control signals, which would otherwise travel away from the receiver, back toward it, allowing us to control the turntable. There are discrete circuit implementations for either IR receiving and transmission that we could have built, but because of the aforementioned reflection constraint and the limited space atop the turntable for additional hardware, we opted to purchase a pre-built IR transceiver. We used the Adafruit IR Remote Transceiver, operating at a typical IR frequency of 38 kHz. From here, discovering and logging the remote commands is simple.

To discover the remote's commands, we attached the output pin of the transceiver to our provided logic analyzer. To properly visualize the length of most infrared communication protocols, we configured the logic analyzer to trigger on the falling edge of a signal, taking twenty thousand samples with a 250 kHz sampling frequency. Provided with the logic analyzer are pre-written decoders for the major infrared protocols: NEC, RC-5, and RC-6. As a note, triggering on the falling edge is appropriate if we believe the adopted remote protocol was NEC, which it was. Using this method, the command to rotate counterclockwise is visualized below, with its complementary decoding as well:

From this, we discovered and logged the following commands, all sent to address 0x00:



Fig. 2. Demodulated IR Remote Signal for Rotating Counterclockwise

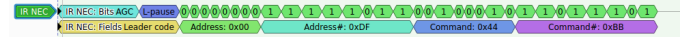


Fig. 3. Decoded NEC Protocol

Start/Stop: 0x42, Rotate Clockwise: 0x44, Rotate Counterclockwise: 0x41, Speed Up: 0x0A, Slow Down: 0x43. Now, these commands must be sent from the FPGA instead of the remote.

#### B. Infrared Protocol Emulation

This section follows from a discussion of the NEC IR Protocol found in [1] and [2]. We have each of the commands. Now, to properly emulate the NEC Infrared Communication Protocol, we must remember two things: one, the demodulator performs the logic inverse of the received signal, and two, it also removes the 38 kHz carrier signal characteristic of this protocol. So, when looking at the two images above, while it may seem that 1 is sent by having a time period where the signal is high longer than it is low, the inverse is true: 1 is sent using a short period of carrier and a longer period of silence. To clarify, high periods are not simply 1, but instead a 38 kHz signal alternating between 1 and 0 with a duty cycle of 50%. Now we know the pitfalls. At its simplest level, NEC sends three general periods of signals, some of which can be further subdivided. First, there is a period of 9 milliseconds of the 38 kHz carrier, then 4.5 milliseconds of silence. After this burst and silence, the address and command bits are transmitted, specifying to which device the command must be sent and what that particular device should do upon receiving it. There are 8 address bits, then their logical inverse, followed by 8 command bits, similarly followed by their logical inverse, for a total of 32 address and command-related bits sent. Sending a 1 is the carrier for 562.25 microseconds, followed by 1.6875 milliseconds of silence—three times the length of the carrier—and sending a 0 involves transmitting the carrier for 562.25 microseconds and then silence for the same amount of time. Once the address, inverse address, command, and inverse command have been sent, the remote sends a final carrier wave lasting 562.25 microseconds. Using this information, we can create the following module, shown below:

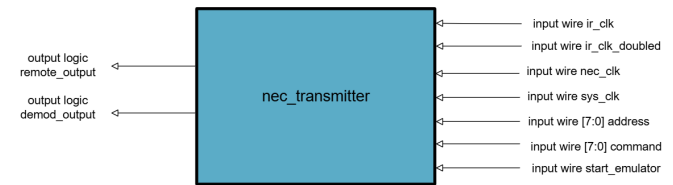


Fig. 4. NEC Emulator Module

Above, the `nec_transmitter` takes in four clock signals: the 100 MHz FPGA system clock, an infrared clock at 38 kHz, another infrared clock at 76 kHz, and finally the NEC clock at roughly 1.78 kHz. If we were to clock the transmission logic at 38 kHz and only trigger on the rising edge, the `nec_transmitter` output would not be a 38 kHz carrier but instead a uniform 1. The 76 kHz clock's inclusion is to allow our module to faithfully output a carrier wave at 38 kHz while simplifying the SystemVerilog logic by not requiring us to trigger on both the rising and falling edges of the 38 kHz clock. The NEC clock is purely for debugging purposes, but it is very useful nonetheless.

A state machine for the NEC module can be seen below with annotated transitions:

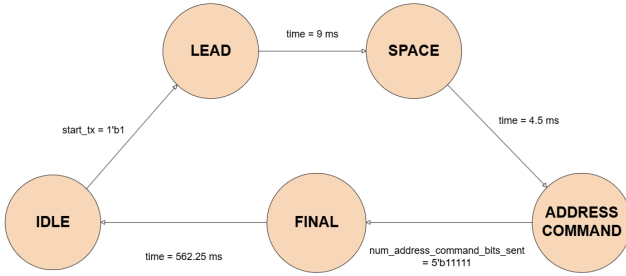


Fig. 5. NEC Finite State Machine Diagram

Because there are five possible commands that we could want to send from the FPGA, but only four buttons, and with one reserved for the system reset, we cannot simply have a button per command. Instead, we encode commands using the first three switches, which can encode 7 possible commands, then press `btn[0]` to send the command. The command is only sent on the rising edge of the debounced button input, so each button press only sends the command once.

The entire IR control module has been implemented in hardware and is functional. A demonstration of it working is here: simple IR control demonstration.

#### IV. IMAGE PROCESSING AND STITCHING - FEYI

As the camera rotates, images are collected at different angles and can be combined into a variety of views: panoramic, birds-eye, and, if time permits, a 3-D surround view.

##### A. Panoramic Transformation

Say that an image,  $I_i$ , is collected at an angle  $\theta_i$  relative to the arbitrary zero angle  $\theta_0$ . To transform into the coordinate system of the image taken at  $\theta_0$ , we construct a homography matrix composed of both the camera's intrinsic matrix and the rotation matrix. As discussed in [3], a homography is a composition of the image intrinsics, rotation, tilt, and translation matrices, such that when applied to  $I_i$ , it is projected into the coordinate system of our reference image, say  $I_r$ . For the panorama, our camera is positioned at the center - this means that the only change between captured images is the rotation angle around the y-axis. Thus, the rotation part of the homography can be written as:

$$R_i = \begin{bmatrix} \cos(\theta_i) & 0 & \sin(\theta_i) \\ 0 & 1 & 0 \\ -\sin(\theta_i) & 0 & \cos(\theta_i) \end{bmatrix}$$

Furthermore, the camera intrinsic matrix  $K$  has the following form:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

, where  $f_x, f_y$  are the focal lengths of the lens in pixels for the  $x$  and  $y$  directions, and  $c_x, c_y$  are the dimensional centers of our output image.

We are using the OV5640 wide-angle lens, with an active array size of 2592 by 1944 pixels, a focal length of 1.79 millimeters, and an image area of 3673.6 microns by 2738.4 microns. Our final homography transform for  $I_i$  is  $H_i = K R_i K^{-1}$ . For homographies, because we are projecting onto a planar surface, the larger the angle of rotation  $\theta_i$ , the more warped and distorted the transformed image becomes. Thus, for this view, we are creating a 180-degree panorama stitch with smaller angle increments. Because of this, and because sin and cosine values involve lots of decimals, we needed a way to not only represent negatives but also decimals in a high-precision manner. To accomplish this, we chose to represent our coefficients in a two's-complement fixed-point format, with 1 sign, 12 integer bits, and 19 fractional bits. To meet timing constraints, we pipeline the computation of these matrices over many cycles.

In practice, the values being transformed are the coordinates  $(x_{in}, y_{in})$ . However, due to sizing requirements to get the full panorama to be able to fully stitch the panorama, we introduce additional transformation and clipping modules. At the end of the transformation pipeline, our  $(x_{in}, y_{in})$  maps to the correct  $(x_{out}, y_{out})$  for the respective image, which is then stored in a buffer, as shown in Figure 5.

We opt to limit each image in the panorama to have a size of 320 by 180 pixels—four times downsampling—such that when stitched together and overlapped onto a 1280 by 720 frame, sharpness and clarity are preserved.

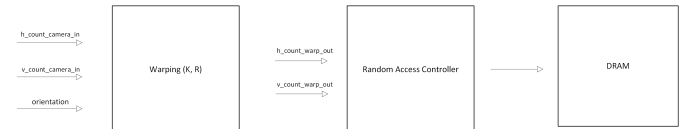


Fig. 6. Collection Pipeline

Because the images are collected at different times, they are stored in DRAM, and post-processing transformations are made after.

##### B. Blending

Given a set of multiple images transformed, if we were to stitch them together, there would be visually apparent seams between them. A common technique used to remedy this is

called  $\alpha - \beta$  blending. For two images, say that they overlap at a common pixel, with data values  $p_1, p_2$ . The final pixel written will be  $\alpha \cdot p_1 + \beta \cdot p_2$ , for specifically chosen values of  $\alpha$  and  $\beta$ . With our smaller rotation degrees, there may be more than 2 images that overlap. Thus, we opted to just do a simple averaging of all pixels.

With all of our images stored in the correct output sizing, we needed to scan over all pixels in the output frame, retrieve the correct slots in storage, and blend them.

A state machine for the blend module can be seen below:

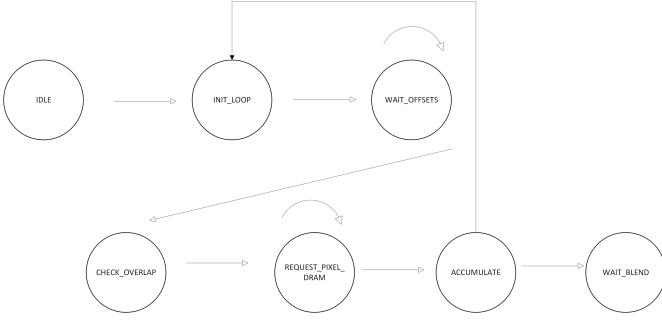


Fig. 7. Blending Pipeline

Following the homography transformation, the images after an offset transformation is applied to them, to ensure that most of the data that the image stores is kept. During collection, we wanted to store each image's bounding box, which the offset calculations were derived from. We retrieve these offsets, then check if our current pixel is within the bounds. If so, we know that the current image contributes to the eventual final pixel value. We retrieve it from DRAM and repeat until all images have been scanned. Then, we blend them together, and output a final value to be stored in the final buffer.

## V. EVALUATION - MICHAEL AND FEYI

The most important part of our design was performing homographies, and this module required extensive pipelining. This module operated on the pixel clock at 74.25 MHz, not the 200 MHz camera clock. But, because we were doing matrix multiplications with entries specified using 1 sign bit, 12 integer bits, and 19 fractional bits—requiring 64 bits to not potentially overflow the multiplications—each computation took significant time relative to our clock. So, we pipelined to a single computation per clock cycle, whether that be a multiplication, addition, shift, or negation. Chief among the challenges was that after performing a homography upon the corners of an input image, the  $x$  and  $y$  coordinates needed to be divided by the  $h$  coordinate. Dividers are variable latency modules, so we needed to wait for the results of eight divisions: one for  $x$  and one for  $y$  for each of the four corners. In simulations, depending on the homography angle, calculating the final homography matrix took on the order of two thousand to three thousand clock cycles. However, once calculated, each coordinate required four clock cycles at 74.25 MHz to compute its  $x'$  and  $y'$  from the input  $x$  and  $y$ .

We down-sampled images before performing the homography, taking only 320 by 180 pixels. So, assuming 2000 clock cycles to compute the homography, and then 57600 pixels multiplied by four clock cycles per pixel, each output image required  $2000 + 57600 \cdot 4 = 232400$  clock cycles to compute. Given that there are 74.25 million clock cycles per second, each image transformation only required roughly 0.3% of the time before sending another command to the turntable to begin rotating again.

From the post-build resource utilization reports, we are using 1 RAM18E1 and 52 RAMB36E1 for a total of 52.5 BRAMs used. In addition, we are using 67 DSP48E1, mostly within the `calculate_homography`.sv to speed up the 32-bit multiplications and 64-bit additions necessary for multiplying matrices together. In terms of optimizations, had we decided to use fewer decimal bits of precision for the homography module, then the FPGA would have used fewer DSP blocks to speed up these computations, and might, for low enough bit sizes, have been able to route them on the normal FPGA fabric. Of course, that might have also downgraded the output quality of the homography. In terms of BRAM utilization, we instantiated BRAMs because we struggled to have random-access writes to DRAM.

Our system meets timing by 0.538 ns of slack. This was after transitioning the homography calculations from the 200 MHz camera clock to the 74.25 MHz pixel clock with a clock domain crossing FIFO in between.

Interacting with DRAM proved challenging. Because we needed random reads and writes, the latency of our module increased a lot, since a single write now issues 2 memory requests which have to finish. However, because the NEC transmitter requires second-long pulses, the increased latency was not expected to have an effect on our system. We attempted to create a controller that accomplished this before eventually deciding to default back to BRAM storage, which gives us random access storage inherently.

Our system was designed to handle any situation where surround view imagery was required—such as in car vision and surveillance. While in practice, these systems run with multiple cameras, we were limited to 1 camera due to our FPGA constraints. We were unfortunately not able to meet any of our minimum, ideal, or stretch, due to the difficulty of interacting with hardware such as DDR3 and the linkage between each individual module. While we were able to get most of the individual components working, such as being able to transform images, and ability control the IR turntable, as well as being able to blend pixels, linking these together proved challenging.

Given the fact that we were able to implement the necessary algorithms and that the majority of our modules were functional in isolation, the complexity of the project stemmed more from creating the necessary infrastructure to link the modules. Because of this, we speculate that the delta between commitment and the next goals was smaller than reaching the commitment itself.

## VI. INSIGHTS AND FUTURE IMPROVEMENTS - MICHAEL AND FEYI

Frankly, we severely underestimated the difficulty of both writing data to and reading data from the DRAM. With many different modules running on different clocks, coordination between them proved to be much more challenging than we first thought. When first discussing our project, we envisioned the most complexity in performing the homography, and while this did prove challenging, writing non-consecutive pixels to the DRAM and having them read onto the HDMI output was much more challenging. Thus, we defaulted back to writing data into BRAMs, and abandoning DRAM storage.

Another frustration was in emulating the infrared-controlled turntable commands. Initially, we erroneously triggered the logic analyzer to capture on the rising edge, not the falling edge. So, the automated decoder told us that the turntable utilized the Philips RC-5 infrared communication protocol, which was surprising because we originally predicted that the cheap turntable would use the cheapest protocol: NEC. But after implementing Philips RC-5, the commands did not work. So, thinking that the turntable used a mutated version of some infrared protocol, we switched to simply trying to reproduce the sequence seen by the receiver. Of course, this was flawed in that the receiver removes the carrier wave necessary for proper infrared communication, and we also needed to realize this. Armed with this understanding and then triggering on the falling edge instead of the rising edge, the turntable indeed used the NEC infrared communication protocol, which the logic analyzer could also nicely decode. So, what should have been a relatively easy task to implement took much longer than expected.

As for future improvements, we could first understand our physical hardware a bit better. When ordering the turntable, we did not consider its size, assuming that the FPGA would easily fit on top. But, it does not. So, the idea of rotating the FPGA with the turntable was flawed from the beginning. Next, we were trying to have a totally wireless setup, but not realize that both the HDMI transmitter and receiver required power supplies to function, adding wires which made rotation virtually impossible. Next, we should have spent much more time understanding the nuances of the provided DRAM memory modules and the ways in which we would need to modify them for our specific application. We allocated a lot of effort trying to enable pixel-level reads/writes, and in hindsight, switching back to our BRAM default at an earlier point in time would have increased our chances of accomplishing our goals. Armed with these insights, our future endeavors would go much smoother.

## VII. DIVISION OF LABOR - MICHAEL AND FEYI

Michael researched and implemented the infrared communication logic, and also wrote the SystemVerilog of the homography module. Feyi implemented the blending module and researched the image homography and stitching methods. Both Michael and Feyi worked together to handle the integration of the different modules at the top level, including

progress towards the DRAM controller that was not able to be completed. Each team member created the visuals related to their main part of the project.

## VIII. LINK TO CODE

Here is a link to our current code: git repo.

## ACKNOWLEDGMENT

Thank you to Professor Joe Steinmeyer and Yoland Guo for their continued support and feedback on this project. Thank you to Kiran Vuksanaj for writing the Vicoco library and Hasan Yildiz for providing feedback on an earlier iteration of this project.

## REFERENCES

- [1] ADOH, "NEC Infrared Transmission Protocol," Sibotic Wordpress Site (PDF), 28 Nov. 2013.  
Available: <https://sibotic.wordpress.com/wp-content/uploads/2013/12/adoh-necinfraredtransmissionprotocol-281113-1713-47344.pdf>
- [2] SB-Projects, "NEC Protocol," SB-Projects knowledge base, last updated 17 July 2025.  
Available: <https://www.sbprojects.net/knowledge/ir/nec.php>
- [3] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed. Springer, 2022. Final draft, September 30, 2021.
- [4] A. Torralba, P. Isola, and W. T. Freeman, "Foundations of Computer Vision: Homographies\*." Available at: <https://visionbook.mit.edu/homography.html>. Accessed: 26 Nov. 2025.
- [5] A. Torralba, P. Isola, and W. T. Freeman, "Foundations of Computer Vision: Camera Modeling and Calibration\*." Available at: <https://visionbook.mit.edu/homography.html>. Accessed: 26 Nov. 2025.