ECE532
Final Report: Visual Auto Navigation

Yang Chen,
Qiuming Yang,
Xiangyi Pan
Will Zhang

Group 20

12 April 2017

1. Overview

1.1 Goals

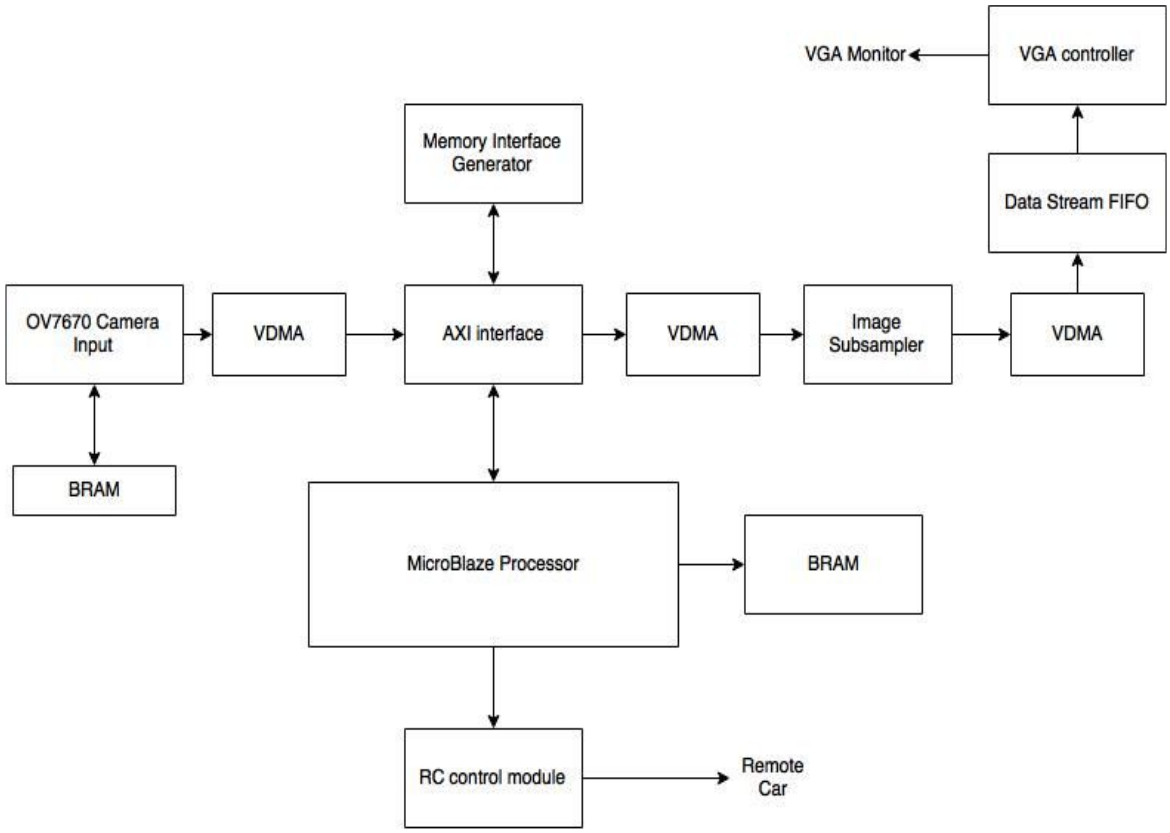
The vision of goal is to develop a system that automatically controls a car to move from a starting point to destination. Input takes form of image captures from images using camera. PMOD camera will be used as the device to capture images, and then sent to FPGA in order to process the necessary information to find the correct angle and the right amount of thrust in order to move towards the right direction.

1.2 Motivation:

The Use case of this system in real life application includes self-driving car. Since most of the components are made using hardware, it can be integrated into real car and used as a replacement to reduce CPU in order to enhance performance.

Another motivation of this project is implementation and practical usage of control theory, the project aims to efficiently use the feedback controller taking images as input and use angle as well as thrust as output.

1.3 Block Diagram

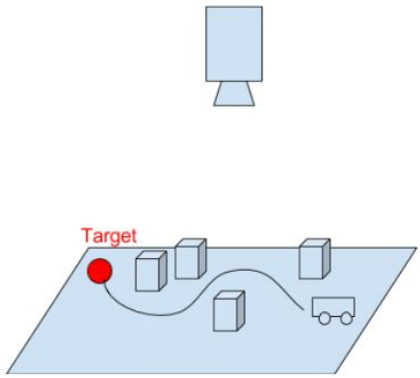


1.4 Brief Description of IP used, modified, created

- Image Subsampler (Custom):
    - To reduce search space and speed up marker detection
- Remote control (RC) car control module (Custom):
    - Provide a simple interface to issue commands to remote controlled car
- Angle Calculator (Custom)
    - Calculate cos value of an angle
- OV7670 Camera Block (Modified from given sample design)
    - Output 320x240 32-bit image (25MHz) from the camera input
    - Generate control signal for the next block
    - Have reference design from piazza resource, customized the design to meet our needs.
- VGA Controller (From previous year design)
    - Convert video stream to VGA signal for monitor
- Microblaze
    - Write registers of RC car control module to drive RC car
    - Calculate angle and distance
    - Control VDMA, Image Subsampler
- VDMA
    - Convert between stream and memory map
- Stream FIFO
    - Cross clock domain
- Memory Interface Generator
    - Interface with onboard DDR to allow larger storage

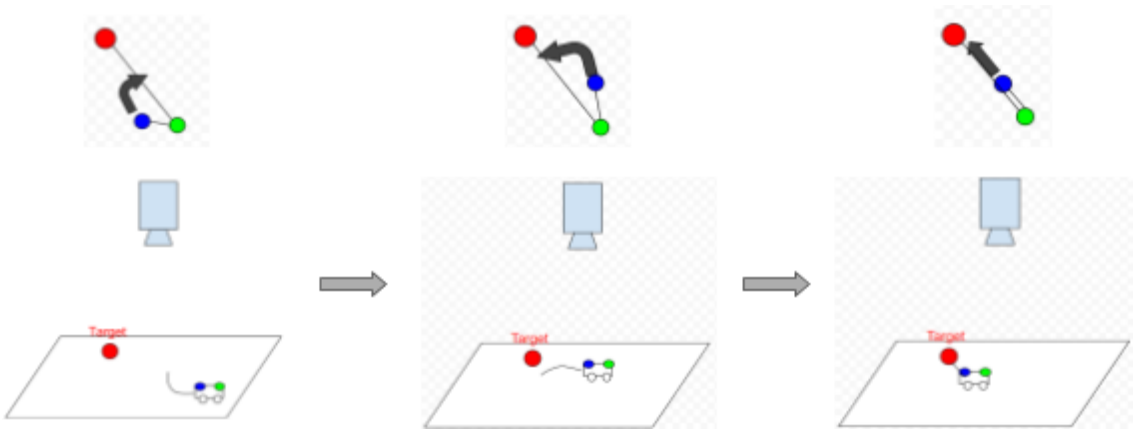
2. Outcome

2.1 Original Plan - Autonomous Pathfinding RC Car



In the original plan, we wanted to implement an autonomous RC car control system. In this system, a camera will feed an image of everything to the FPGA and the FPGA then will calculate a path to that the car can follow. Finally, the car would follow the path to the target without hitting the obstacles in between. The main difference is that we don't have a pathfinding algorithm in our final design.

2.2 Final Product - Visual Auto Navigation



Instead, the car is guided by visual feedback. At each step, the camera will capture the positions of the target and the car based colors. WIth the coordinates of the three points, we are able to calculate the angle formed by the three points with the tail as the vertex. Then the FPGA will move the car accordingly. For example, in the first figure, the head is left to the goal, so the car would turn right. Eventually, the car could reach the target.

2.4 Future Work

The path finding feature that was originally planned would add a lot of sophistication to this project as well as giving the project much more practical applications. Completing this feature is the most obvious next step for any future work on this project.

Aside from this, currently, the remote controlled car has a huge turn radius. To better take advantage of the navigation capability of this project, the remote controlled car should be replaced with one that is much more maneuverable.

Also, currently the feedback control delays are very conservative. With the computational capability and detection accuracy achievable in this project, the delay can be cut down significantly to allow practically continuous operation of the car.

Project Schedule

3.1 Original Milestone Plan:

Milestone #1	● Show basic test bench for image scaling IP core. ● Develop an IP core which takes a 640x480 framebuffer and scale it down to a 64x48 array of pixels by subsampling and averaging the original framebuffer.
Milestone #2	● Show a functional testbench, where an input image is placed into a framebuffer in memory, and a scaled image will be placed in an output buffer.
Milestone #3	● Develop an IP core to generate a software map of the testing topology suitable for input into a pathfinding algorithm.
Milestone #4	● Implement a pathfinding algorithm based on the software representation of the topology generated in the previous milestone. ● Experiment with controlling of RC car using the FPGA.
Milestone #5	● Show individual IP blocks are functional, including control of RC car, processing of image from a 646x480 framebuffer to a software map of the topology
Milestone #6	● Develop an IP core to blit together runtime information such as the calculated path with the input 640x480 video for displaying on the output VGA display.

3.2 Actual Weekly Accomplishment:

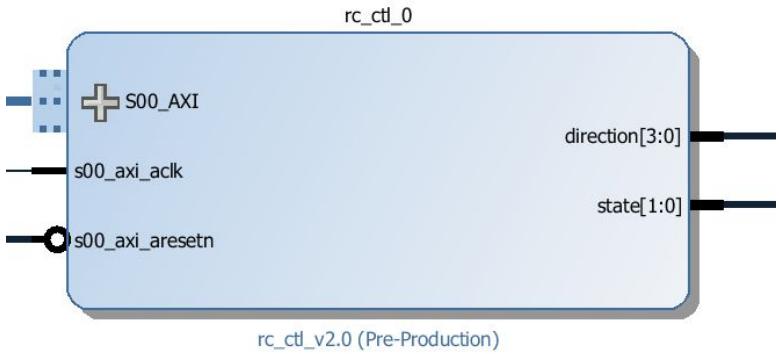
Milestone #1	● Show basic test bench for image scaling IP core. ● Developed an IP core which takes a 640x480 frame-buffer and scale it down to a 64x48 array of pixels by min-pooling the original frame-buffer.
Milestone #2	● Modified the RC controller so that we can use FPGA to control the RC car via GPIO ● Implemented VGA camera and FPGA can drive the camera now ● Hardware implementation of image scaling.
Testbench Demo	● Built an rc car controlling ip which receives rc car direction and distance information and send it to the rc car. And use the Vivado debug core to verify its functionality. ● Enabled the PMOD camera and generate the data stream for scaling.
Milestone #5	● Enabled the rc car to move from point A to point B ● Connected camera with image filter.
Milestone #6	● Drive RC car from point A to point B ● Read test image from memory, pass through filter and store back into memory ● Display filtered image through VGA. ● Identify coordinate of specific color markers. ● integrate filter with camera
Milestone #7	● fixed the camera and enabled it to write image frame to memory. ● integrating the car controller IP to image filter IP. ● integrating the Camera IP to image filter IP.

3.3 Plan changes and reasons

Our actual progress roughly met the requirement in our design proposal. The only difference is that we did not deliver the pathfinding algorithm features. The main reason is that we found that the remote control car is not as perfect as what we expected. The car cannot make a sharp turn and the pathfinding algorithm is hard to implement. We struggled to solve the problem but still did not find the acceptable solutions. Besides, several unfortunate events, such as laptop missing, and poor complexity management are also the reasons that we failed to deliver all the features in our proposal.

4. Description of the IP Blocks

4.1 Remote Control (RC) Car Control Module:



The RC control module provides a simple interface to issue commands to a RC car (e.g. up, down, left, right). It consists of 5 registers.

I. Direction Reg
This register has 4 useful bits representing up (bit0), down (bit1), left (bit2), and right (bit3) signals. Therefore, bit0 and bit1 are sending signals to motor while bit2 and bi3 are sending signals to steering. For example, if bit0 and bit2 were set to high, the RC car will turn left.

II. Count Reg
The value of this register represents a period of time (number of clock cycles) to drive the RC car in a specific direction. For example, if we wanted the car to go straight for 1 sec, we need to set this register to 100M (the board frequency is 100MHz).

III. Control Reg
With this register, it allows us to control the input and output of the module so that we can adjust the input parameters to the module based on the current state. When bit1 of this register is set to high, this module will start sending signals to the RC car. It also allows us to control the car step by step so we can debug the system easily.

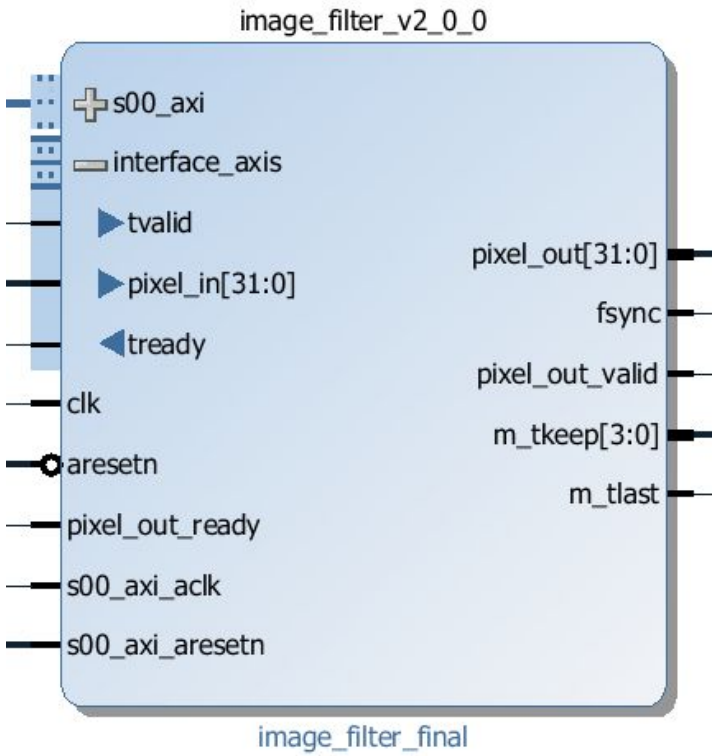
IV. State Reg
In this system, there are two states - IDLE and RUNNING - represented by this state register.

V. PostPreCount Reg

Before starting each step, we want the RC car to be in a fully still state. This count is used to cancel out the effect of inertia. For example, the car would continue to move for another 3 cm after the power (motor) is turned off.. Therefore, we need to keep the direction (steering) for a while after turning off the motor. So this count represents a period of time (number of clock cycles) to just steer the RC car before and after the motor is turned on and off respectively.

With all this registers working together, we are able to send desired signals to PmodCOM with connects to the controller in order to control the RC car. And we also output the state so we can display it on the LEDs for debugging purpose.

4.2 Image Subsampler:

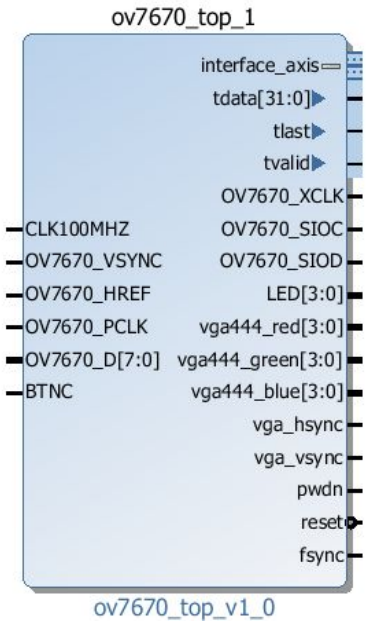


The image subsampler Interfaces with input and output framebuffer through stream interface with 2 VDMA blocks. The interface data width is 32 bit, corresponding to a transfer rate of 1 pixel every clock cycle. Synchronization with VDMA blocks are achieved using the fsync signal.

Internally, the image filter uses 64 flip flops, individually receiving pixels from the VDMA. Once each flip flop has received all pixels in the required subsampling region, the subsampler outputs 1 line of output image after receiving 10 lines of the

input image, corresponding to a scaling ratio of 10. Uses min-pooling as the filtering strategy, meaning the darkest pixel in each sampling region is used as the destination pixel. The image subsampler is controlled by the MicroBlaze processor with a Control Register through AXI-lite interface.The control register consist of just an enable bit, which, when set causes the first fsync signal to be send, starting the subsampling.

4.3 PMOD Camera Module:



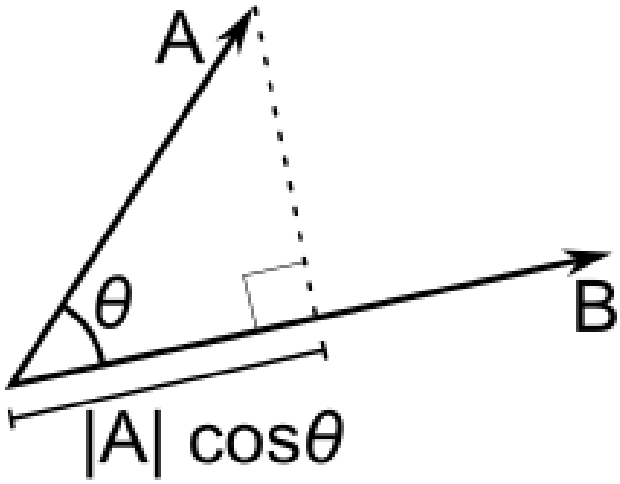
We studied the PMOD camera IP block from the Piazza and used it as our reference design. The new camera IP block now enables tlast, tvalid, fsync signal, as well as the data stream. In our new camera IP block, there are six signals from the external. The CLK100MHZ is driven by the system clock, which is set to 100MHz. The OV_7670_VSYNC, OV_7670_HREF, OV7670_PCLK, OV7670_D[7:0} and BTNC are connected to the CMOS image sensor. These signals are set the same as the reference design. The output data stream ('tdata') is a 32-bit data stream, which contains the RGB signal of each frame. The data has the following format:

$$tdata = [8'b0,\ vgared(4bits),\ 4'b0,\ vgagreen(4bits),\ 4'b0,\ vgablue(4bits),\ 4'b0]$$

Fsync is the signal to synchronous each frame. Tlast is the signal which indicates the end of each line in the frame. It has been set to 480 in our design. Tvalid is the data valid signal which communicates with the VDMA. The other signals are the output of this IP block and they have the same functionality as the reference design.

Reference design link:
https://piazza.com/class_profile/get_resource/ixov7014v1f23d/iyz3c568nc44xz

4.3 Angle Calculator



The angle calculator uses dot product in order to obtain the cosine value of the angle between cart and its destination. The tail end of vectors A and B represents the tail end of the remote car, head of the vector A represents the head of car, and head of vector B represents the destination point.

- The value of the coordinate which represents the x, y coordinate of three points being used are passed into the module.
- Magnitude of A and B is then approximated with finite state machine.
- Approximation method being used is known as the babylonian approximation, it is an iterative method that approaches the exact value of square root as the number of iterations increase
- Magnitude of A, B is calculated using the square root function in order to obtain the real value for it.
- Finite State Machine:
    - Finite state machine is used to account for the approximation,
    - there are three states to the approximation procedure, start, Step1 and Step2 and final.
    - Values are initialized in start state, all parameters are set to 0
    - Counter is incremented every time the procedure finishes each iteration.
    - Upon reaching correct iteration numbers, the magnitude value for A and B are then recorded
    - The cosine value of the angle between vectors A and B are calculated using the dot product value dividing the magnitude values for A and B
    - Finally the cosine value is recorded

The implementation of angle calculator strictly uses hardware, the advantage of doing so is to reduce CPU usage and reduce software overhead in order to improve the performance of the car controller.

5. Description of Design Tree

- **ece532_capstone** - root folder
    - **doc** - contains group report, slide presentation for final demo
    - **src** - contains all project source files
        - **Board_files** - contains configurations files for Nexys4 DDR
        - **Final_integrated_project** - contains final integrated vivado project
        - **fixed_Camera_core** - contains source file of the OV7670 IP core
        - **Image_filter** - contains source files and test bench of image subsampler IP core
        - **Rc_car_control** - contains source file for the rc_car_control IP core
        - **VGA** - contains source file of VGA controller IP core
        - **Angle_cal** - contains source file of Angle Calculator IP core