VISION-BASED LOCALIZATION OF MULTIPLE ROBOTS ON HETEROGENOUS PLATFORM

A Design Project Report
Presented to the School of Electrical and Computer Engineering of Cornell University
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering, Electrical and Computer Engineering

Submitted by:
Moonyoung Lee (ml634)

MEng Field Advisor: Professor Zhiru Zhang

Degree Date: January 2017

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering
Cornell University
Design Project Report

Project Title: *Vision-based Localization of Multiple Robots using Heterogeneous platform*

Authors: Moonyoung Lee (ml634)

Abstract:

With growing popularity in cheaper and more compact microcontroller platforms, multi-robot systems is becoming an increasingly popular area. However, most mobile robotic applications require rigorous processing tasks such as real-time vision and navigation that exceed what popular microcontrollers can process. Equipping each robot platform with higher-end processors would not be scalable due to increase in cost and power consumption. Instead, this paper explores the design of using a single accelerated processing unit that handles the navigation control for all robots. The processing unit leverages heterogeneous system architecture of ARM Cortex and FPGA. The ARM processor handles the main execution flow of the project while the FPGA fabric handles the offloaded video input to process object detection algorithm. The microcontroller on each robot performs motor control according to the navigation commands received from the processing unit. The purpose of this project is to explore the hardware-software co-design of a heterogeneous platform for controlling multiple scalable robots. Example applications would be building site construction or wildfire containment by dispatching fleet of modular autonomous ground vehicles with a drone that captures global view point. Instead of increasing hardware and software complexity on individual robots, a single computing station can process the visual data needed for absolute localization and broadcast navigational commands to all the robots on the field.

This paper introduces the key components of the project in software implementation first, and then discusses the implementation in the heterogeneous platform. The heterogeneous platform resulted in a performance speedup of 52x over the baseline design. The performance is evaluated by the time required to process a single frame before sending the navigation command to all robots. Higher frequency of broadcasting commands directly results in improved robots navigation. The expected processing time for software system is approximately 870 milliseconds while the heterogeneous platform is 17 milliseconds.

Executive Summary

The objective of this project was to implement a vision-based multi-robot control system on a heterogeneous system architecture of ARM Cortex and FPGA. With the ARM handling the overall control flow and the FPGA handling the off-loaded image processing function blocks, the system broadcasts navigation commands to each robot according to the position and orientation detected from the video stream input. By using an FPGA for the video processing, real-time processing and navigation control could be achieved. This project required learning about how to develop an image pipeline in hardware that exploits parallel-structured code, which included pixel programming, filter windows and line buffers, and data transfer from the FPGA to the ARM.

The initial software development implementation using OpenCV on personal laptop was completed as a proof of concept and to rapidly determine design choices. This included finalizing the server-client network, vision algorithm, and robot navigation control. From this initial software benchmark, the robot control was coarse but successfully navigated the robot to the desired goal. The next software benchmark was to implement the similar tested setup to the heterogeneous platform but executing it purely on the ARM. This benchmark did not utilize libraries such as OpenCV or Video HLS for implementing the image processing pipeline. The software benchmark was also successfully implemented and from this benchmark, the robot control was not feasible due to the significant processing delay. The software-hardware co-design resolves this processing delay by utilizing the performance of FPGA, which would provide real-time image processing and ultimately real-time robot navigation capability.

There were unresolved run-time errors that would result in deadlock for the synthesized hardware implementation in the center of mass function block. This issue prevented complete synthesis of the image processing pipeline in which center of mass and corner detect function blocks were not ultimately included for the data captured for evaluation. However, observing the speedup measured from initial function blocks synthesized, it is expected that the full synthesis of the image processing pipeline would maintain, if not increase, the speedup reported for this project, ultimately observing real-time navigation for each robot. This project provided significant software hardware co-design experience for computationally-intensive application of computer vision.

Table of Contents

1. Introduction

1.1 Motivation

An essential component of mobile robot systems is the ability to accurately determine robot's position and orientation relative to its environment. For autonomous mobile robots, using sensors to locate the robot in its environment becomes the most fundamental problem. Many recent research developments use sensors such as sonar, laser range finding, or vision modules with probabilistic algorithms to accurately solve robot's global self-localization problem. However, these solutions typically address localization of a single robot only. Scaling localization techniques independently to each robot in a multi-robot system becomes increasingly costly and complex. Instead, this project shifts the vision sensor module and relevant processing away from the robots to a single processing unit that leverages FPGA.

It is becoming increasingly popular to integrate field programmable gate arrays (FPGAs) accelerators to mobile robotic applications in order to achieve post-manufacturing flexibility, energy efficiency, and performance. This balance of efficiency and performance introduces significant benefits to mobile robotic systems, a domain where significant amount of data needs to be processed real-time while minimizing the battery consumption. Vision based simultaneous localization and mapping (SLAM) has become especially widely-adopted for mobile robot systems. Utilizing SLAM to accurately navigate its environment require significant video processing and filtering schemes. In these applications, FPGA greatly increases performance in processing speed by reconfiguring the underlying hardware to exploit parallel architecture. However, developing FPGA prototypes in RTL language can be very complex and time consuming.

This paper presents a heterogeneous computing platform of mobile multi-robot application that trades off some of the processing performance of FPGA devices for major reduction in design complexity by integrating ARM processor with FPGA fabric to a single processing unit. Example applications of the heterogeneous system outlined in this project would be building site construction using scalable multi-robots. Global visual inputs of the field site can be processed on the accelerated processing unit to broadcast control commands to the multiple modular robots on the field. Such system design would reduce cost and complexity involved in the multi-robot localization problem.

1.2 System Overview

In order to make the multi robot control scheme more scalable, this project clearly divides the processing unit from the robots. Rather than having each robot process its own position and orientation towards the goal destination, each robot only listens to the commands broadcasted from the processing unit, which visually determines each robot's navigation. The HD camera input is passed to the processing unit via HDMI cable, processing unit performs object recognition algorithms to determine position and orientation, processing unit broadcasts unique navigational commands for each robot through wireless access point, and the robot performs motor control according to the given commands to reach its destination. The system flow of this project is illustrated in the figure below.

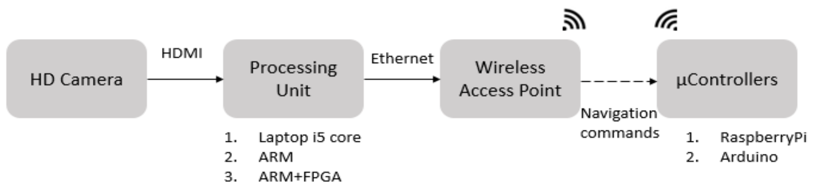

Figure 1. System flow of the described project

The processing unit is interchanged throughout the project for evaluation purposes. The first implementation was done using the Laptop i5 core as the processing unit for rapid development purposes. In order to better gauge the performance improvement utilizing the hardware/software co-design, the baseline design of the project utilized only the ARM A9 Cortex as the processing unit. Lastly, the alternative design of the project utilized both the ARM and FPGA on the Xilinx Zynq board that leverages synthesized accelerated functional blocks.

To summarize, the benchmarks are implemented in three parts:

- laptop with Intel dual core i5 processor (development)
- Xilinx ZC702 board with dual core ARM Cortex-A9 processor (Baseline)
- Xilinx ZC702 board with dual core ARM Cortex-A9 processor and FPGA (Alternative)

The performances of across the three benchmarks are evaluated in terms of frequency of robot commands broadcasted. It is expected the performance of the alternative design will have significant speedup over the baseline and the development benchmarks.

2. Robot Platform

2.1 Robot Design Decision

The design decisions for the robot platform revolved around maximizing the navigation capability of scalable robots with minimal hardware. In contrast to the traditional implementation of robot navigation, the robots built for this project do not utilize a camera, inertial measurement unit, and encoders that provide corrective feedback capability. Instead, these robots only utilize their Wi-Fi modules to depend on the commands broadcasted from the processing unit to continuously correct their navigation as the HD camera processes in real-time. The project was aimed to target popular programming devices of RaspberryPi v3 Model B board and Arduino-based Intel Galileo, Edison boards.

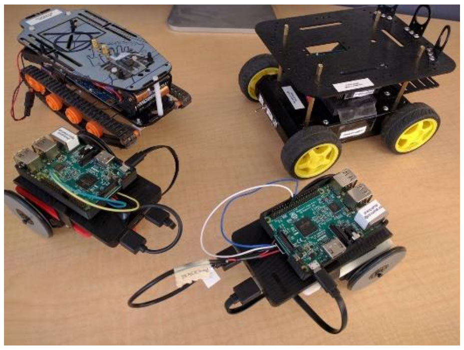

Figure 2. Robots utilizing Arduino and RaspberryPi platform

The popular RaspberryPi board provided the flexible peripherals, Linux-based OS, and built-in Wi-Fi module for quick development. To summarize the robot's simple software, upon boot-up, the RaspberryPi board checks for socket connection, waits until receiving valid commands sent in byte packages, and handles motor control according to the speed and rotational commands received from the main processing unit. In order to further increase the scalability of the robot module, the robots required only single power-source, single control board, and 3D printed chassis.

The navigation commands, composed of speed and direction bytes, are sent from the processing unit. These navigation commands are structured independently of the robot platform to maintain versatile and scalable design by being able to control both RaspberryPi and Arduino based robots.

2.2 Assembly of the Robot

The robot parts used for the RaspberryPi is provided by Scout on 3D print open-source community on Thingiverse. The robot chassis contains holes that can be used to mount standard servo dimensions. The marble and O-rings were ordered to fit accordingly to the provided CAD dimensions. The wheel and robot chassis was 3D printed from Cornell University's Rapid Prototyping Lab.

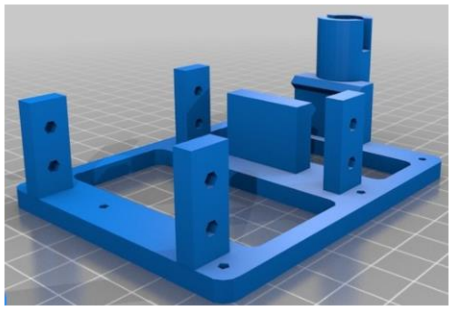

Figure 3. CAD of the 3D printed robot chassis

To maintain scalability for multiple robots, the robot was designed to minimize cost and assembly time by requiring only single battery-source, microcontroller board, and servos. The assembly process only requires 3D printing, mounting the listed components, and splicing the microUSB to connect the servo and RPi pins. The Linux OS provides ease of connection to the wireless access point.

Item	**Description**	**Qty**	**Cost**
Raspberry Pi Board	Model 3 for built-in Wi-Fi	1	$36
microSD card	32GB class 6 or above	1	$11
Continuous Servo	Full rotation, 3 pin, 3.3V signal	2	$26
Battery Pack	5V dual USB ports	1	$13
Robot Frame	3D Printed	1	
O-ring	Friction for wheels (2-⅛" OD, ⅛")	1	$6
Marble	Center wheel for robot (½")	1	$9
microUSB cord	Will required splicing	2	$7
	Total cost		$108

Table 1. Bill of Material for the robots

2.3 Robot Software

In order to ease development with multiple robots, a shell script is launched upon boot up of the RPi that launches the main python script. The main python script creates a client socket that listens to the predetermined static IP address of the server. The client listens for 1 byte packets, which is then translated into one of the 4 unique navigation commands. The 4 commands that control PWM of the 2 servos from the RPi are translated into one of the following: goLeft, goRight, goStraight, Stop. For the processing unit to broadcast these commands at above 50 Hz allow for the robot to seamlessly navigate to the desired goal destination.

3. Software Implementation

3.1 Software High-level Description

The main processing unit is implemented in three parts: on a laptop with Intel dual core i5 processor with 2.2 GHz processing frequency, on the Xilinx ZC702 board with dual core ARM Cortex-A9 processor with 667 MHz frequency, and on the ZC702 board with ARM and FPGA. The first implementation on the laptop is without the hardware implementation and developed as the functional level model using the popular OpenCV library in C++. The second implementation is on the ARM processor only to handle pixel by pixel handling of the video frames without utilizing OpenCV library. Lastly, the third design is an extension of the ARM implementation in which all image processing functions are synthesized as hardware to offload video processing on the FPGA. The report discusses the project software implementation of baseline design in Section 3 and the hardware implementation of the alternative design in Section 4.

The control flow in all three design implementations can be summarized as follows: the processing unit receives the video input, performs object detection algorithm to determine position and orientation, and sends navigational commands to the wireless access point, which is then broadcasted to all the robots on the field. With higher frequency of command updates, the robots' navigation to the desired destination can be processed in real-time and significantly improved. The third design implementation with heterogeneous platform explores this alternative design. The control flow of the project is illustrated below.

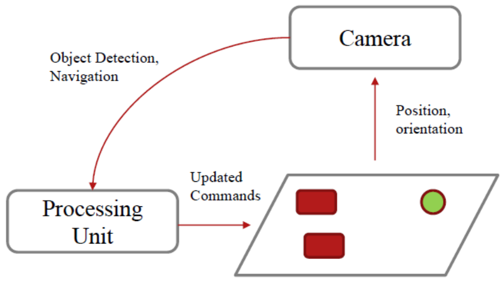

Figure 4. Control flow to control multiple robots from the processing unit

3.2 Baseline Design Server-Client Network

The processing unit is responsible for establishing a simple client-server network, robot tracking from the video input, and sending updated navigation commands to the wireless access point. A simple server-client network was implemented using the Windows Winsock 2 library and TCP/IP to transmit packets between client, the Raspberry Pi's (RPi), and the server, the processing unit. The server-client network was setup by creating the socket for IPV4 with TCP protocol on the laptop, binding to the server socket, and then accepting any connections made during initialization stage on run-time. The command broadcasted to each RPi was either stop, steer left, right, or straight. These commands are mapped to a 2 bit values packed into 1 bit, which then sends to the socket with additional parameter of which client socket to broadcast to. The RPi client then performs motor control from the received byte.

3.3 Baseline Video Processing Algorithm

The robot tracking vision algorithm was implemented using the OpenCV libraries to leverage the abstraction of the matrix operations. After the client and server connection has been established, the camera video input stream is processed to navigate the multiple robots to the desired destination, indicated by a tennis ball. The image processing pipeline of performing matrix operations is summarized in the figure below.

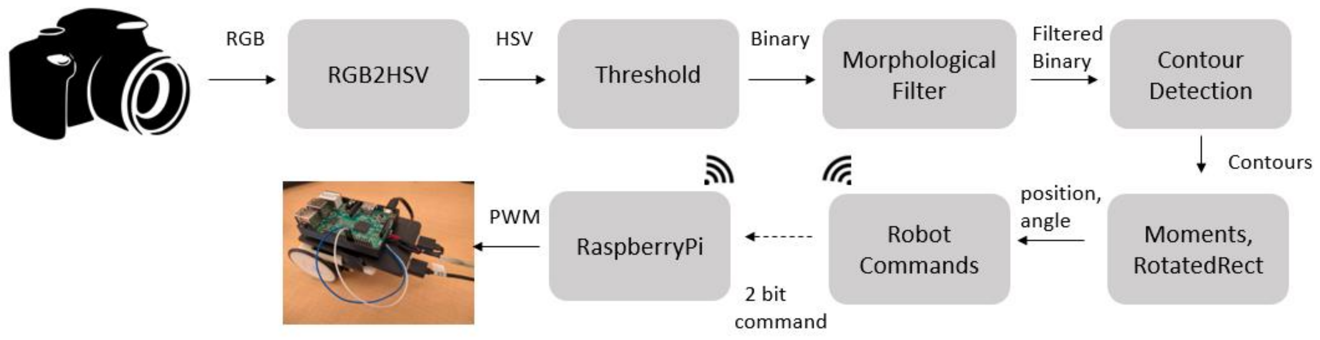

Figure 5. Software image pipeline for robot control

In order to easily distinguish the ball from the robots, we process the raw captured RGB input into the HSV color scheme. Operating in HSV color scheme provides significant benefit over the RGB color scheme such as isolating the desired color into light and value intensity. Bounding the lower and upper bound range of a particular color value is narrowed to one variable rather than three variables of RGB in raw input color scheme. This isolation also provides robustness to fluctuation in lighting condition. The next matrix operation is to floor or ceil pixel values into a binary matrix is within the desired HSV color bounds for black robot and neon green tennis ball color.

The threshold outputs corresponding to the ball and robot color are then de-noised with a morphological filter that erode and dilate the binary matrix with a window size of 5x5. The morphological filter resulted in a better contour detection over alternative filtering schemes such as traditional median or

Gaussian filter with a 5x5 window. The resulting binary matrix from eroding and dilating function maintains the edge while filling in the non-connected values in the surrounding body.

From both the ball and robot threshold matrix that has been morphologically filtered, the center position is determined by using the 1st order or the centroid moment. The weighted average x, y coordinates of the binary matrix can be determined by the equations below.

$$\bar{x} = \frac{M_{10}}{M_{00}} \qquad \bar{y} = \frac{M_{01}}{M_{00}} \tag{1}$$

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y) \tag{2}$$

As pixel iterates through the entire frame, the raw image moments M_{ij} is computed by summing the non-zero pixel intensity *I(x,y)* for row and column. *I(x,y)* would be binary pixel values due to thresholding the matrix. M_{00} holds the total count of non-zero pixels, M_{10} holds the sum of non-zero pixel column values, and M_{01} holds the sum of non-zero row values according to Equation 2. Dividing the computed sum as shown in Equation 1 yields the desired centroid, or center of mass represented by $\bar{x}$ and $\bar{y}$.

In order to track individual robots on 2D plane, a robot class is created to store ID, previous position, current position, and orientation. In order to distinguish between the two robots, the current center of mass of the robot is compared to the previous position. If this difference is within specified range of pixel, then this robot is marked as the same robot from the previous frame. To prevent situations where this difference in positions could be accidentally interchanged between two robots, the algorithm can place a specified radius around each robot to alter navigation commands if the two robots are in trajectory for imminent collision.

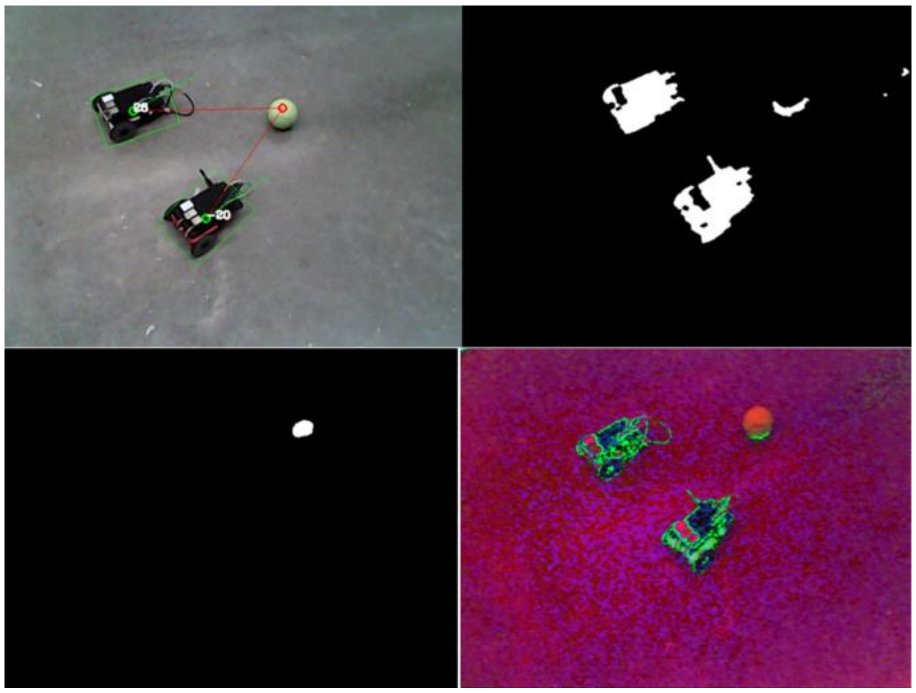

Figure 3. Position and orientation detection toward desired goal

The orientation, captured in angles, is also determined using the OpenCV library RotatedRect which inscribes the detected object with area and angle. Linear lines are extended from centroids of each robot to the destination. The line intersection between the extended line and each robot's heading creates an angle difference that is used to determine the updated command to each robot. In the next frame, this vision algorithm and command update are repeated to coarsely navigate each robot to the desired destination. The image frames captured to visualize the matrix operations are shown in Figure 3. Each robot's heading is captured in the green line, the desired trajectory to the goal is captured in the red line, and the angle between these two lines are used to determine robot's navigation commands.

3.4 Robot Command on Raspberry Pi

Once the center of mass and orientation is determined, these variables can be mapped to one of the 2 bit robot commands of left, right, straight, or stop. The arctan of the 2 points associated with yMin and xMax extrema is used to determine the robot's orientation. Then the arctan of the 2 points associated with the center of mass of the robot and the goal is used to determine the desired heading to reach destination. The signedness of the difference in the two angles either map to left or right command. If the angle is less than ±10 ° then the robot would drive straight until the absolute distance between the robot and the goal is less than specified radius in pixels.

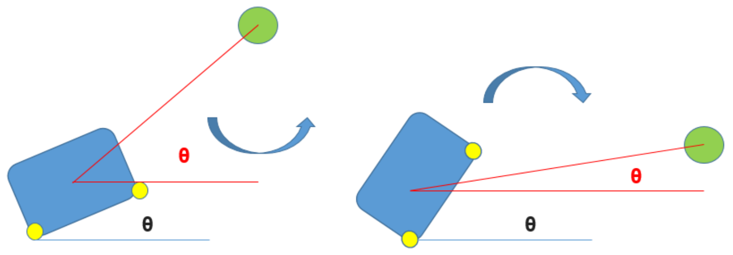

Figure 4: Robot command determined by center of mass and corners returned.

4. Hardware Implementation

4.1 Hardware High-level Description

In the heterogeneous processing unit, Dual core ARM Cortex-A9 is the host processor handling main execution flow of the project and FPGA fabric is the video accelerator handling the offloaded processing-intense task such as tracking the multiple robots from the camera input. The project improves ease of development by leveraging Xilinx Vivado design suite for High-Level Synthesis (HLS), which automates the design process of interpreting the algorithmic behavior of high level programming languages to corresponding RTL. For this project, Vivado HLS is utilized to synthesize hardware accelerated function blocks such as RGB2YCbCr, median filtering, center of mass detection, corner detection, and YCbCr2RGB. To integrate the ARM and FPGA platforms, we utilize the SDSoC (Software Development System on Chip) design suite to develop the hardware-software co-design. SDSoC is the wrapper to the Xilinx Vivado design suite which easily enables heterogeneous computing to exploit the familiarity in ARM Core and acceleration of FPGA. The SDSoC tool abstracts the interface between the Processing System (PS) and the Programming Logic (PL) by appropriately creating drivers to access AXI video stream data. The project setup in hardware is shown in figure below.

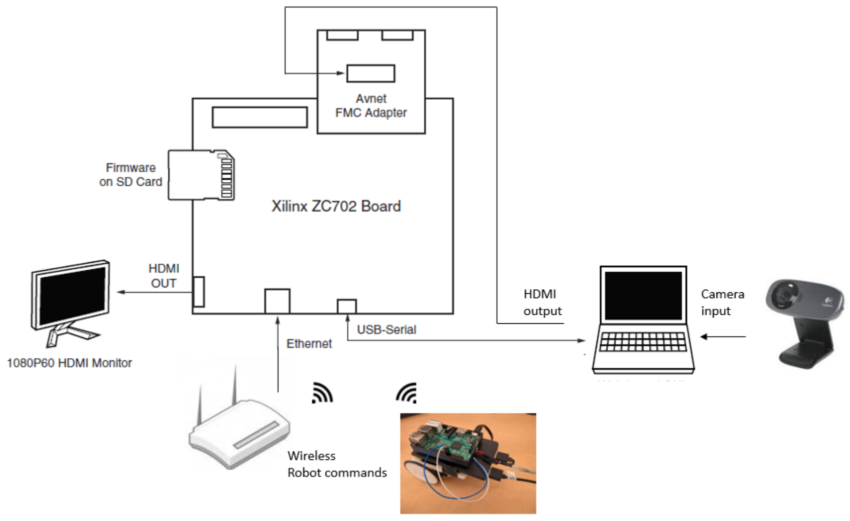

Figure 4. Project setup of the hardware implementation

The project flow is similar to the software implementation, with the processing unit now being substituted by the Xilinx ZC702 board. The laptop is only used to provide the HDMI output from the USB camera input. The HDMI monitor is to display the resulting pixel matrix from the image processing pipeline. For this project, the monitor displays the red, blue, and green detected output from the two colored robots and the tennis ball. The hardware implementation is different than the software implementation because it computes operations pixel by pixel rather than utilizing abstraction provided by the OpenCV library for the entire matrix computation. Due to this additional complexity, the hardware implementation simplified the robot detection by marking them with distinct color of red and blue.

4.2 Hardware System Architecture

This project utilizes the frame-buffer streaming architecture for video processing, in which pixel data is stored in external memory. The video frame buffer stores pixel data in the PS DDR3 memory, allowing Linux on ARM to access the frames via the AXI Video Direct Memory Access (VDMA). Video input/output chain writes/read frames into a circular buffer with space for 3 frames. The frame buffer allow for the software and hardware accelerator to access video input and output.

The supported video format for video frame is 1080p HDMI with 60 frames per second. Each pixel represents a 4 byte value (x, R,G,B). Video frames are streamed 2211480 (1080 x 2048 stride) for input and output 1 pixel at a time. We will then use the AXI4 streaming protocol to communicate pixel data, where each line of video pixels is an AXI4 packet (beginning pixel and end pixel of each row of frame marked with flag). The system architecture of the hardware implementation is shown in the figure below.

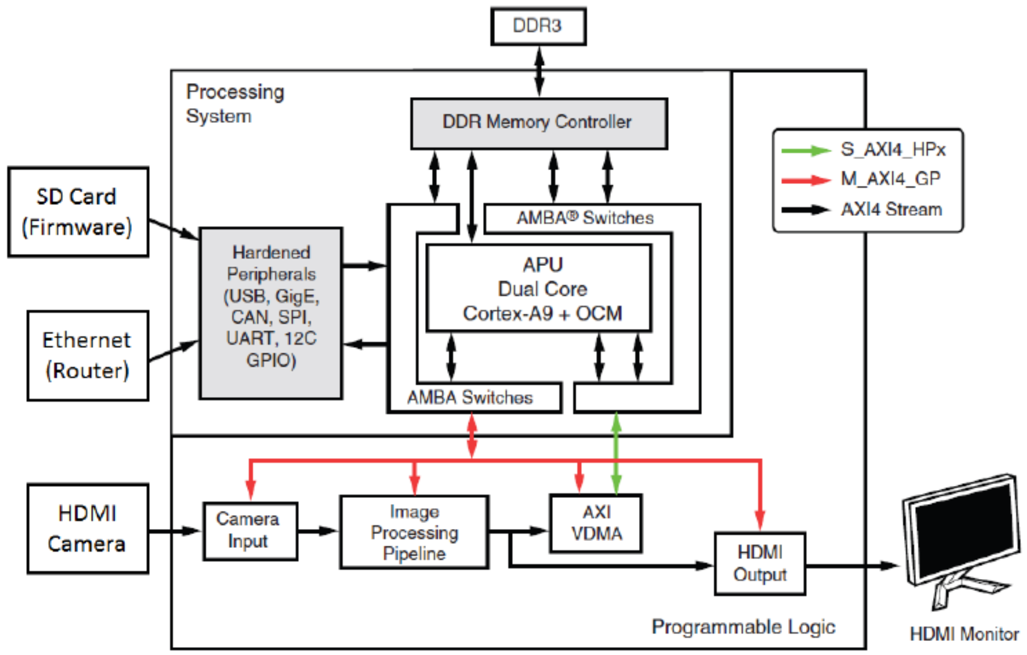

Figure 5. Block diagram of the PS and PL interface for HDMI video stream [REF]

The software sync function *setCVC_TPGBuffer* allows for automatic access to next input/output frame when it's available. Additional benefit of leveraging the SDSoC tool is that it determines each function call and maps to the desired accelerator. In addition, SDSoC also handles inputs to each accelerator generated from other hardware functions. The code layout for the project is as summarized below.

Code Structure:

Src: Top level code *main.c*

- Links virtual memory to physical memory space for video frame buffer
- Setup TCP/IP network
- Calls series of synthesized image processing function blocks
- Receives 2 center positions, 8 corner points
- Determine robot commands from position and orientation
- Broadcast robot commands

Hw: Image functions to be synthesis for accelerator

- Input RGB2YCBCR
- Threshold by Color (red and blue for robot, green for ball)
- Median filter 5x5 window
- Center of mass detection
- Corner detection
- YCbCr2GRB output

4.3 ARM Cortex 9 (Processing System)

The software and hardware distinction is made clear in the project setup. The main function is the software on the ARM that handles the control flow. It links virtual memory to physical memory space for video frame buffer, sets up TCP/IP network for the server-client environment, calls series of synthesized image processing function blocks, and then determines the robot command to be broadcasted through the wireless access point.

In order to establish communication between the ZC702 board and the Raspberry Pi robots, a server-client network that utilizes Wi-Fi is implemented. The ZC702 board and the Raspberry Pi boards on the robots are connected to a wireless access point so that they are in the same local area network. Sockets are used so that the application running on the ZC702 board could be connected to the python programs running on the Raspberry Pi robots. In our design, the ZC702 board is considered to be the server while the robots are considered as clients so that it would be easier for us to send out different commands to robots separately. In order to facilitate the connection of the multiple robots, the Raspberry Pi launches a shell script upon boot up to execute the python script to connect to the server and wait for motor control directions.

4.4 Xilinx FPGA Fabric (Programming Logic)

The hardware implementation is significantly more complex than the software implementation because OpenCV library was no longer utilized for image processing. The SDSoC environment did not provide compatibility with the OpenCV equivalent of HLS Video library, so the hardware implementation resorted to limited pixel by pixel stream programming.

In order to achieve high performance in the image processing pipeline in the FPGA, we utilize Vivado directives or pragma to exploit the static HDMI frame size as well as instruction/data level parallelism in our code. The image processing pipeline for the FPGA is shown as below. To facilitate development and debugging of pixel by pixel programming, an external monitor is used to display the state of the processing pipeline.

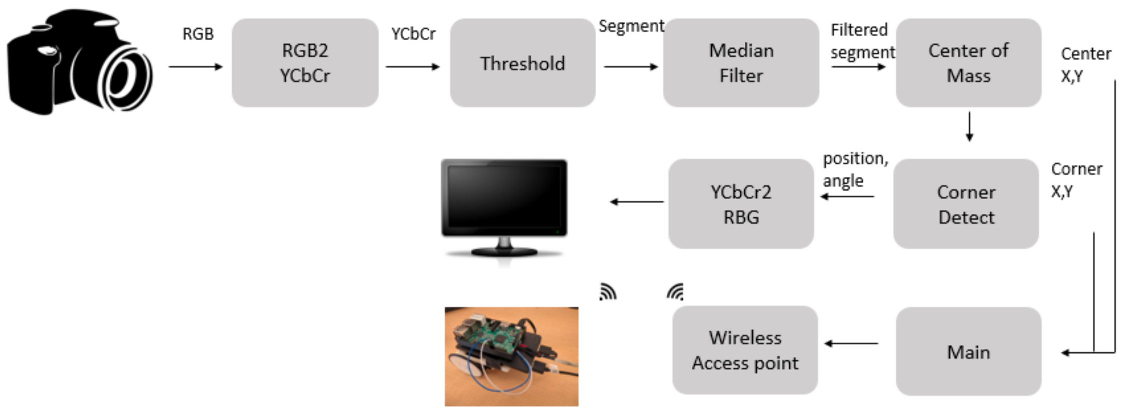

Figure 6: FPGA image pipeline to the HDMI output and wireless access point.

4.4.1 RGB2YCbCr Conversion and Threshold

Similar to the software baseline design, the initial function block in the image processing pipeline was to convert the input RGB pixel values into a color encoding that is robust against varying conditions. For the alternative design, instead of the HSV scheme, the YCbCr was utilized to easily separate the red and blue colors. Because no convenient tool of image processing library was available with the project setup, the alternative design compromised the complexity of the computer vision algorithms used. Instead of segmenting the robot by contours as done in the baseline software implementation, each robot was tagged with different color to easily segment one from the other.

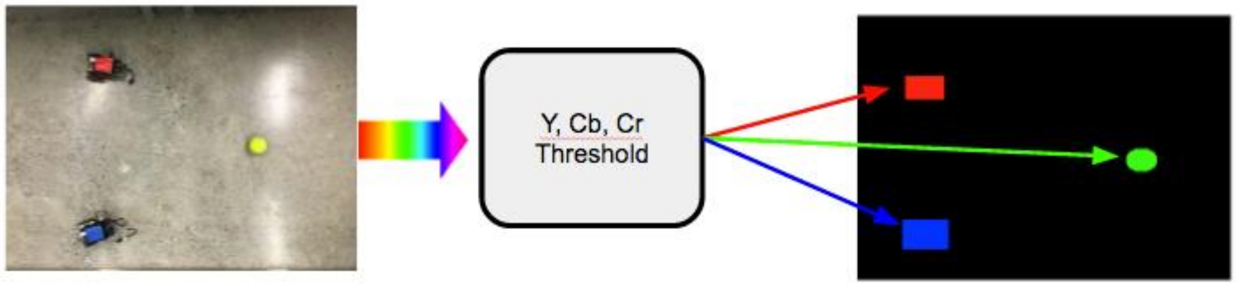

Figure 7: YCbCr color segmentation from a 4 byte RGB input image to a 4 color output image.

4.4.2 Median Filter

To improve robustness from noise, the image pipeline included a median filter that effectively reduces noise and also preserves edge information. A median filter is an operation that checks all the pixels within a certain size window around a center pixel. Below is an illustration of a median filter's operation using a window size of 3x3. As seen in the figure, the center pixel value's 8 neighbors are collected and sorted in numerical order. Then, the median value of the sorted list is noted. This median value replaces the original center pixel value as seen in the figure where 10 is replaced by the median value of 4. Therefore, by median filtering a video frame, noise caused by outlier values can be removed.

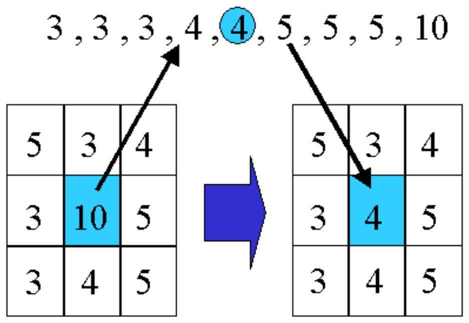

Figure 8: Median filter example with a window size of 3x3.

A median filter with window size of 9x9 was used in the image processing pipeline's second stage to remove unwanted noise from the video frame. In contrast to other image processing function blocks, median filter is more computationally expensive because it requires additional nested for loop checking in the window for each pixel. To fetch a load instruction from memory for all 8 neighboring pixels for all pixels in a 1920x1080 frame could be a significant bottleneck to the performance. To avoid this, the filter utilizes HLS provided ap_window and ap_lineBuffers. As the median filter window slides across the entire video frame, many of the neighbor values will be repeated. Instead of reloading these values from memory, the neighboring pixels are stored in local memory and for a quicker read of pixel values. The only value that needs to fetched from memory would be the pixel in the bottom-right window, which would be a first instance of reading that pixel value. By using line buffers, only new pixels are shifted into the filter's window on each iteration, which greatly improves performance of the filter.

4.4.3 Corner Detection

In contrast to the baseline design, the orientation of the object detected cannot be simply determined by utilizing the RotatedRect function provided by OpenCV. In order to determine the robot's heading, the image processing pipeline includes corner detection of the thresholded frame. This function block is responsible for calculating the corners' coordinates of both the red and the blue robot and passing them to the main function by storing the determined extrema to the pointer in the virtual memory so that ARM can read values calculated from the FPGA. Since there are four corners for each rectangle-shaped robot, a total of eight corner coordinates are to be calculated.

To calculate the coordinates of one robot's corners, extrema of x and y-axis values within the robot object have to be determined. Each corner's coordinate has either the maximum or the minimum value on x and y axes, as long as the object is a convex polygon with four corners (apices) or a concave polygon with four corners pointing outwards itself. This principle applies to the tested robots, since both are detected as rectangular shapes which have four corners. As a result, robot's corner coordinates can be represented by (xmin, ymin), (xmin, ymax), (xmax, ymin), and (xmax, ymax). The function block scans through the frame and records the current pixel's coordinate if it finds a new extrema.

To add robustness to the corner detection scheme, a temporal median filter is included in this function block. It records the corner coordinates' history which contains the last 11 sets of corner coordinates and outputs the median of the 11 to the main function. The median set is obtained by looking for the median of each of the eight coordinates for all determined extrema, which is the result set of coordinates that needs to be returned to ARM for determining robot control commands.

5. Results

The alternative design of using both SW and HW outperformed the baseline design of purely SW, as well as the laptop development benchmark. Performance is measured by the total amount of time elapsed to process one image frame. By processing frames at shorter time intervals, the ZC702 board can send higher frequency of robot drive commands to achieve more accurate navigation. Qualitatively, it was observed that the baseline design processed image frames at every 2.8 seconds, which had too much of a time delay to control the robots. The robot would drive out of the frame, update its command from couple seconds prior, return back on screen, and then stop. The path trajectory of the robot drive for the baseline design in shown in blue line below. With faster processing with Intel i5 laptop processor, an adequate but inefficient navigation was observed because the robot was overshooting every command as shown in green line. The expected result for the alternative design is shown in red line, which closely follows real-time navigation.

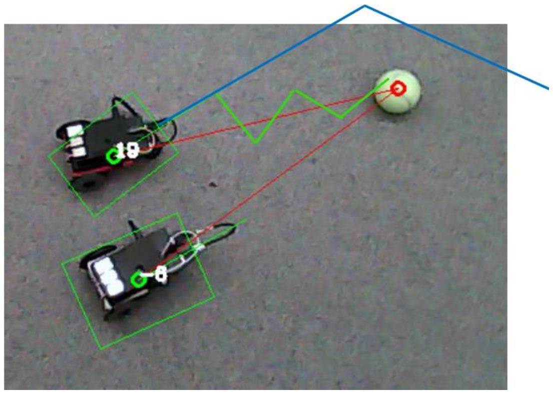
Figure 12: Performance of robot navigation according to the three benchmarked implementation

	Time Elapsed per frame (milliseconds)			Area Utilization			
Application	**Software**	**Hardware**	**Speedup x**	**LUT**	**DSP**	**FF**	**BRAM**
Color Conversion	**892**	**17**	**52.5**	**24958**	**29**	**32068**	**35**
Threshold	**870**	**17**	**51.2**	**24787**	**24**	**31871**	**35**
Center of Mass	**1037**	*****	*****	**32822**	**23**	**43765**	**39**
Median Filter, Corner Detect	**44253**	*****	*****	**41800**	**24**	**55039**	**69**

Figure 13: Summary of the performance and area utilization of both design implementation.
**Data not collected as explained below.*

The hardware was synthesized incrementally as more function blocks or functionality to the image pipeline was added. As expected, a significant speedup was observed in how quickly the image frame can be processed when utilizing the FPGA. The peak speedup measured was 52x from thresholding the different colored robots. The median filter was also independently synthesized and resulted in a speedup due to utilizing line buffers as described above.

However, measurements for the complete image processing pipeline was not captured due to unresolved deadlock issues in synthesized hardware. Although there are no collected data to measure the speedup for the complete image processing pipeline, the speedup observed is expected to increase when including the median filter and other function blocks, so that the robot drive commands would be broadcasted at a rate at least 52x speedup to achieve real-time control. This performance speedup comes with trade-off in area utilization and design complexity. As shown in table above, with added functionality to the image processing pipeline, we see increase across area utilization for LUT, FF, and BRAM, while the DSP usage stays unaffected.

5.1 Hardware Optimization

There were some limitations initially encountered using the targeted was Xilinx Zynq ZC702 board, both in not meeting timing requirements and exceeding available LUTs. With better handling data types, such as reducing each pixels to 8 bit ap_uint after segmenting rather than maintaining the 4 byte unsigned int throughout the entire image pipeline, these area issues were resolved. To increase performance, the project utilized specified pragmas to ensure pipeline II of 1 for every for loop, specifying dataflow in the top function block, as well as specifying FIFO interface between input/output of the synthesized blocks. Initially, the code structure reflected poor structure for parallel pipeline processing such as having long sequence of dependent instructions after the nested for loops for pixel streaming. This resulted in timing violations that was eventually resolved by minimizing instructions outside loops.

With these optimizations, it can be concluded that the alternative design of utilizing the FPGA would achieve real-time navigation without too much trade-off in area utilization. The baseline design is simply not feasible for any real-time application with HD image processing.

5.2 Challenges

The main challenge encountered was the hardware implementation using the SDSoC tool. Although the tool provided significant benefit of abstracting the RTL language from the user, this abstraction proved to be difficult when the synthesized hardware would display unexpected results in contrast from the original software C/C++ code.

When function blocks were built and the software version was run, the system displayed both to HDMI monitor and to the terminal the expected values. Upon booting from the SD card, the program would initiate the network connection and the output frames displayed on the external monitor with corresponding colors filled in and centers of masses marked, along with corners. However, it did not turn out to be the case for the hardware. When executing the hardware version, the program was stuck in a deadlock after the two robots were connected to the board and no output frame was displayed. Since the synthesized hardware blocks cannot print out values or display pixel values until the end of the image processing pipeline, it was difficult to fully gauge the state of each function block. To debug, small incremental functions had to be removed and resynthesized, which resulted to be a very tedious and not a deterministic method of solving the problem.

The initial debugging was to isolate the deadlock issue between the image processing pipeline and the network setup. Even after removing the network function, the hardware program got stuck immediately since execution. Afterwards, the image processing pipeline had to be isolated to each block to see where the failure point would occur. The deadlock issue was then isolated down to being caused by writing / reading array values in a 1 depth FIFO, which is the default depth upon instantiation. This issue was resolved by cleverly storing the needed values into the first few pixels stream as recommended by Professor Zhang. Lastly, there was another cause of a deadlock where multiple function blocks were both writing to the same pointer to memory, disrupting the sequence of image pipeline process to also cause deadlock issues.

6. Conclusion

The objective of this project was to implement a vision-based multi-robot control system on a heterogeneous system architecture of ARM Cortex and FPGA. With the ARM handling the overall control flow and the FPGA handling the off-loaded image processing function blocks, the system broadcasts navigation commands to each robot according to the position and orientation detected from the video stream input. By minimizing the computation time for image processing, the project observed a speedup of 52x for the synthesized partial portion of the image processing pipeline. By using an FPGA for the video processing, if the complete image processing pipeline could be synthesized for hardware, real-time processing and navigation control could be achieved. This project required learning about how to develop an image pipeline in hardware that exploits parallel-structured code, which included pixel programming, filter windows and line buffers, and data transfer from the FPGA to the ARM.

There were unresolved run-time errors that would result in deadlock for the synthesized hardware implementation in the center of mass function block. This issue prevented complete synthesis of the image processing pipeline in which center of mass and corner detect function blocks were not ultimately included for the data captured for evaluation. However, observing the speedup measured from initial function blocks synthesized, it is expected that the full synthesis of the image processing pipeline would maintain, if not increase, the speedup reported from this project, ultimately observing real-time navigation for each robot. This project provided software hardware co-design experience for computationally-intensive application of computer vision.

7. Acknowledgement

The author would like to thank Professor Zhiru Zhang and Steve Dai from Computer Systems Laboratory at Cornell University for their constant guidance in developing the project. Their insight and design recommendations have proved greatly helpful for developing this project. In addition, I am thankful for all members who participated in improving this project in the Zhang's Research Group as well as in the ECE5775 final project.

8. References

[1] eyeRobot – Heterogeneous Robotic Platform for Real-Time Computer Vision, *Mohit Modi, Sravya Chinthalapati*

[2] Controlled iRobot with Intel Galileo, *Qiukai Lin*

[3] A Printable tri-bot frame, *Scout* https://www.thingiverse.com/thing:13042

[4] Bruce in the Box, *Julie Wang*
http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/s2014/jsw267/html/html/

[5] ZC702 Evaluation Board for the Zynq7000 XC7Z020 All Programmable SoC
http://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850zc702evalbd.pdf/