Final Group Report
ECE532 Digital Systems Design

Self-Driving Delivery Robot

Team #5
Sheran Cardoza
Shahrooz Ghayouri
Fyyaz Khan

Date of Submission
April 8th, 2019

Table of Contents

1. Project Overview

This section states the motivation behind our project and presents the project details including goal and block diagrams.

1.1. Motivation

We were inspired by autonomous drones that can be instructed to deliver goods to destinations. A swarm of drones can be programmed to perform deliveries at scale to multiple hard-to-reach destinations. Such a delivery system is highly efficient and can greatly ease logistics.

As part of our project, we will emulate such a remote-controlled drone system that can be programmed to perform an automated task. Our project will include all aspects of a drone system, including remote control, self-driving capability, and destination tracking.

1.2. Project Goals

The goal of our project is to build a robot that can be instructed to autonomously drive towards a given destination marked by an LED.

1.3. Block Diagram

Figure 1 shows an overview of the block diagram of our system.

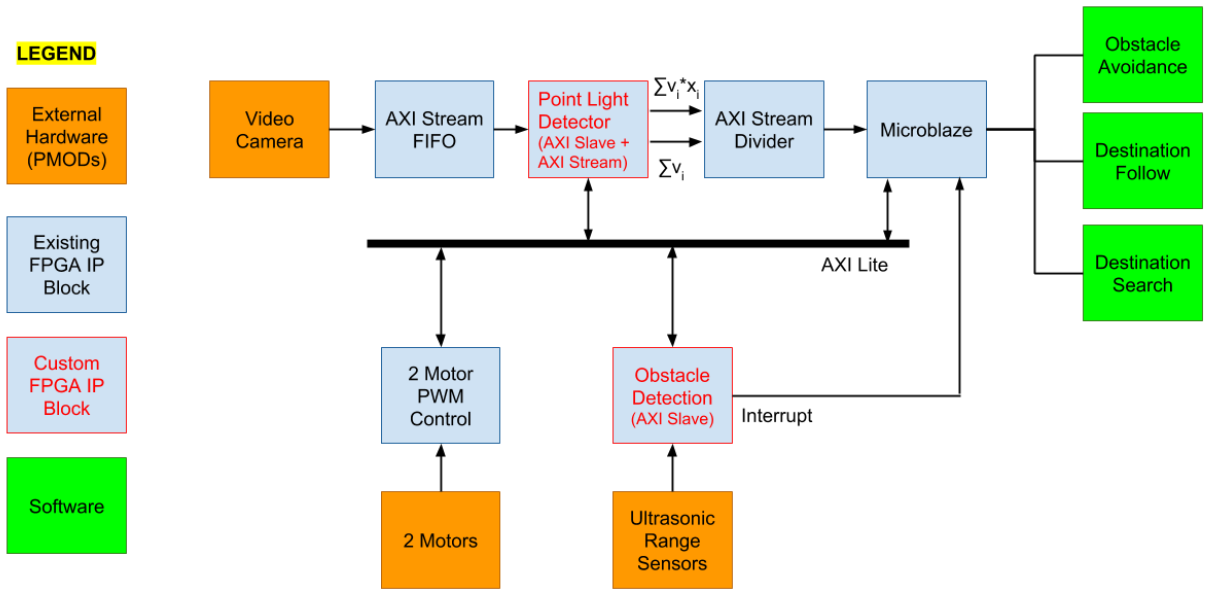

Figure 1: **System block diagram**

The video camera output is fed to the Point Light Detector over AXI Stream. The Point Light Detector analyzes the incoming video feed and detects the position of any LED in the image. The Microblaze processor drives the motors and polls the Point Light Detector to search for an LED and steer the robot. The Obstacle Detector monitors the Ultrasonic Range Sensor PMODs to detect the presence of any obstacles in the vicinity and interrupt the processor.

1.4. IP Blocks

The custom IP blocks used in our design are:

- *Point Light Detector* - outputs the position of any LED in an image
- *Obstacle Detector* - monitors Ultrasonic PMODs and interrupts processor if obstacles are detected

IP blocks borrowed from Vivado's IP library are:

- *Microblaze processor* - processor to execute motor control logic based on LED position and obstacles
- *AXI Stream Divider Generator* - Configurable Divider IP with AXI Stream Interface
- *AXI Stream FIFO* - Data FIFO with AXI Stream interface
- *AXI Lite* - bus for the processor to interface with motor drivers and custom IP blocks

The PMOD IP blocks used in our design are:

- *PWM IP* - AXI Slave configurable PWM generator
- *MAXSONAR PMOD IP* - interfaces with the Ultrasonic Range Detector PMOD

IP blocks borrowed from past projects are:

- *video_in_ip* - used to convert the video camera output to AXI stream data

Section 4 describes these IP blocks in more detail along with their sources, and also details the software components of our design.

2. Project Outcome

This section presents the final results of our project and potential areas for improvement in the future.

2.1. Results

Our original goal was to create a self driving robot that would be able to receive a command over wifi that specifies a specific target color. The robot would then use a camera to lock on to the destination marked with this target color, and steer towards it. Any obstacles along the way would be detected by a set of ultrasonic sensors and avoided. The robot would also be able to run through a routine to search for a marker outside of its field of view.

Feature	Status
Receive Commands over Wifi	Not Working
Detect and Steer towards LED	Working
Avoid obstacles	Working

The major feature we are missing from our original design is the ability to receive commands over WIFI. The reason for this is we ended up spending much more time than we expected getting the LED tracking working. This left us no time to integrate the WIFI system into our final design.

We were able successfully create a robot that could detect a colored object and drive towards it. One change from the original feature was that we limited the tracked object to red, green or blue, rather than allowing arbitrary colors. In addition to this the design was more sensitive to noise than we had hoped.

The obstacle avoidance HW and SW routines worked as well as we had expected as the robot was able to stop before hitting and object, and correctly change course to navigate around it.

2.2. Areas for Improvement

Destination Detection

One of the biggest issues with our current design is its sensitivity to noise in a room. It is very easy to throw the robot off course by introducing objects to the area similar in color to the marker being detected. While tightening the thresholds may help filter this noise out it makes it much more difficult to detect the destination marker, especially under changing lighting conditions. A more dynamic thresholding method may make the destination detection more robust. Alternatively a tracking technique that takes into account the shape of the destination marker may be more useful.

Ultrasonic Sensor Noise

Our use of two ultrasonic sensors in close proximity introduces some noise to the signal. This manifests as false object detections every so often. One possible solution would be to only activate a single ultrasonic sensor at a given time when it needs to take a reading.

Motor Controller

The biggest issue with the motors is their consistency. No two motors are exactly the same. To solve this problem we manually multiplied the final duty cycle of one of the motors by some constant factor to slow it down to match the speed of the other motor. This value was

determined by testing the motors. A more robust approach to this would be to add support for motor feedback. This would allow you to set the exact target RPM of the motors and have that be achieved through a feedback loop.

Search Routine

When the LED goes out of view of the camera the robot needs to be able to move around to find it again. The current search routing does a full 360 turn to see if the marker is anywhere around it. If it is not found this way (the LED may be obstructed, or around a corner) then it picks a random direction to travel to and tries again. This method is fairly random. There may be more efficient techniques to searching for the destination. For example keeping track of the last known direction it was in, or mapping the surroundings.

3. Project Schedule

Milestone 1	
Original tasks proposed	i. Obstacle detection IP block in simulation. ii. Range sensor in hardware. iii. Wifi IP block in hardware.
Actual tasks completed	i. Range Sensor set up in hardware (Set up to poll). ii. WIFI IP block set up in hardware
Evaluation	Both these tasks were untested at this point as we did not have out PMODs yet.

Milestone 2	
Original tasks proposed	i. Obstacle detection IP block + Range sensor in hardware. ii. Wifi IP block + Communication from server to Microblaze and vice-versa in hardware. iii. Video camera + Double buffering in hardware.
Actual tasks completed	i. Set up interrupt hardware and AXI interface for Obstacle detection IP ii. WIFI Hardware + client/server software iii. Video camera + output video to VGA using VDMA and TFT controller iv. Motor controller + software drivers set up (for single motor) v. Assembled Robot
Evaluation	Original task i. partially complete. Went ahead of schedule and got the motor controller working as well as the robot assembled.

Milestone 3	
Original tasks proposed	i. Point Light Detector IP block in simulation. ii. Car direction control software. iii. Obstacle avoidance software.
Actual tasks completed	i. Point Light Detector IP software simulation created ii. Point Light Detector AXI Wrapper created iii. Point Light Detector Hardware Module written (only tested in sim) iv. Obstacle detection IP completed (untested)
Evaluation	Obstacle detection IP from milestone 2 completed. Car direction control software and obstacle avoidance software not complete. On the other hand went ahead of schedule for the Point Light detector IP and got it partially done in hardware.

Milestone 4	
Original tasks proposed	i. Physical robot built. ii. Microblaze accelerating, braking, and steering the robot.
Actual tasks completed	i. Point Light Detector IP completed in hardware. Modified to use AXI Stream. ii. Camera interface modified to use AXI Stream iii. Microblaze software for robot control done
Evaluation	Physical robot was already built in milestone 2 (ahead of schedule). We had made some changes to our design at this point (pivoting from writing pixel data to memory, to using an AXI Stream Interface)

Milestone 5	
Original tasks proposed	i. Robot moving + Obstacle avoidance. ii. Separately demo Point light detector IP block in hardware.
Actual tasks completed	i. Point Light Detector integrated with Camera and tested ii. Main software routines written, integrates light detection + motor control iii. Obstacle avoidance integrated but buggy. Not connected in software
Evaluation	Mostly on track for this milestone. Obstacle Avoidance integrated in HW but not on the software side, thus it couldn't control the robot at this point.

Milestone 6	
Original tasks proposed	i. Robot moving towards target when it finds one.
Actual tasks completed	i. Robot moves towards target when it finds one (somewhat buggy still) ii. Obstacle avoidance software completed but not integrated
Evaluation	Had the expected tasks completed but still buggy at this point. Needed more tweaking.

Milestone 7	
Original tasks proposed	i. Integration of everything.
Actual tasks completed	i. Integrated everything except for the WIFI system.
Evaluation	Project complete and working. Did not have time to integrate WIFI into the project.

4. Description of IP Blocks

This section details all the IP blocks used including software components.

4.1. Custom IP blocks

The following IP blocks are custom written to achieve the project's functions.

4.1.1. Point Light Detector

IP block meant to calculate the x,y position of the destination marker of a certain color (red, green, blue). It does so by calculating the average of the (x,y) coordinates for each pixel weighted by the value of its color channel being tracked.

$$x_c = \frac{\sum_i x_i * v_i}{\sum_i v_i}$$

x_c is the average x position of the target color in the frame.

x_i is the x position of pixel i in the frame.

v_i is the thresholded value of the target color channel

Note: the division is not done within the Light detector IP, the divisor and dividend of the final result for a frame are exposed on two AXI Stream interfaces.

The thresholds for a color are R_max, R_min, G_max, G_min, B_max, B_min.
The min is the minimum threshold of the target color to accept that pixel. The max is the maximum value of the non target color channel to accept. To accept a pixel the target color channel must be above <Target Color>_min and the other two channels must be below their given max threshold values. A pixel that does not pass the threshold has v_i set to 0.

There is also a threshold on the final value of the dividend (The total thresholded target color values). If the dividend is below the given threshold we assume the destination marker is not visible, thus not found. We signify "Not Found" by setting x_c to 0 (by setting the numerator to 0).

Pixel values are streamed into this IP through an AXI Stream interface and the sums are accumulated as they come in. When all the pixels of a frame have come through the accumulated divisor and dividend are send to the divider through the two master AXI Stream interfaces. The Accumulators are then reset for the next frame.

Interfaces

AXI Slave - This interface is used to configure the Detector block. The thresholds are set here as well as the target color.

Slave Register (each is 32-bits)	**R/W**	**Usage**
0	R/W	(Unused)

1	R/W	Target Color:
		`Red = 0`
		`Green = 1`
		`Blue = 2`

2	R/W	Color Thresholds:
		`R_max = slv_reg2[4:0]`
		`R_min = slv_reg2[9:5]`
		`B_max = slv_reg2[14:10]`
		`B_min = slv_reg2[19:15]`
		`G_max = slv_reg2[24:20]`
		`G_min = slv_reg2[29:25]`

3	R/W	X position (Unused)
4	R/W	Y position (Unused)
5	R/W	Min Total (Divisor) Threshold

AXI Stream Slave - Stream of pixel data from camera. Each 32bit data entry contains the R,G,B values of the pixel.

Data is packed as follows:

data[23:19]	**data[15:11]**	**data[7:3]**
Red	Green	Blue

Note: The green channel is actually 6 bits but we drop the least significant bit to match the other channels.

AXI Stream Master - Dividend of LED x position for a frame ($\sum_i x_i * v_i$)

AXI Stream Master - Divisor of LED x position for a frame ($\sum_i v_i$)

The two master stream interfaces are intended to connect to an AXI Stream Divider (We used the Xilinx Divider Generator IP block)

Testing Procedure

To test that the AXI Slave interface had been implemented correctly a testbench was created using a AXI VIP block to send AXI requests to the Point Light Detector block. We verified register writing by sending a write to a register and then reading it.

To test the AXI Stream interfaces we used an ILA block to monitor the transactions on the AXI Stream Slave and the AXI Stream Master. We verified that the AXI Stream protocol was being followed by our IP. This task was done in hardware rather than simulation because it would be hard to simulate the camera in our testbench as we don't fully know its behaviour, and hardware testing would be simpler than creating a behavioural model of the camera.

Finally the block was tested in hardware by moving an LED in its field of view and checking that the x, y coordinates changed depending on whether it was on the right or left of the camera.

4.1.2. Obstacle Detector

The purpose of the Obstacle Detector IP block is to monitor the two Ultrasonic PMODs and interrupt the processor if any obstacles are detected in the vicinity. This frees up the processor to perform other tasks by not having to continuously poll the Ultrasonic PMODs.

It contains an FSM that controls polling the Ultrasonic PMOD. The FSM initially contained a reset state which was eventually removed due to issues with global reset. The final FSM contains only one state wherein the IP constantly polls the Ultrasonic PMOD. In this state (called *POLL_MASONAR_PMOD*), the IP continuously issues read commands to the MAXSONAR PMOD IP to read its slave register. A formula is used to convert the read data into obstacle distance in inches. From experimentation, these distances can range from 5 inches to 255 inches.

It also contains another FSM that maintains the current obstacle state, i.e. whether there's no obstacle nearby (*OBSTACLE_NOT_DETECTED*), or there's an obstacle in the vicinity (*OBSTACLE_NEARING*), or an obstacle in the vicinity has been cleared (*OBSTACLE_LEAVING*). An obstacle is considered to be in the vicinity when it crosses a threshold set on the obstacle distance. This threshold can be configured by the processor through software by writing to the IPs slave registers. Upon reset, the FSM begins in *OBSTACLE_NOT_DETECTED* state. When an obstacle is in the vicinity, it transitions to *OBSTACLE_NEARING*. When the obstacle is cleared, it transitions to *OBSTACLE_LEAVING* and then back to *OBSTACLE_NOT_DETECTED*. These states are intended to prevent spamming the processor with interrupts and also inform the processor when the obstacle has been cleared. However when testing on hardware, this FSM was unreliable and instead once the processor is interrupted, the processor itself takes care of monitoring the status of the obstacle.

<u>Interfaces</u>

2 AXI Masters - These two master interfaces contain the aforementioned FSMs and communicate with the slave interface on the MAXSONAR PMOD IPs to read the obstacle distances.

AXI Slave - This slave interface contains the slave registers that the processor can read and write to. The following is the register mapping.

Slave Register (each is 32-bits)	R/W	Usage
0	R/W	AXI clock frequency (Hz) needed for obstacle distance calculation formula
1	R/W	Threshold for obstacle distance below which it is considered to be in the vicinity of the robot
2	R	FSM state info informing the processor whether the obstacle is still in the vicinity or it has just left
3	R	For debug: obstacle distance from first Ultrasonic PMOD
4	R	For debug: set to 1 if obstacle is in vicinity, detected by first Ultrasonic PMOD
5	R	For debug: obstacle distance from second Ultrasonic PMOD
6	R	For debug: set to 1 if obstacle is in vicinity, detected by second Ultrasonic PMOD

Testing

This IP block was challenging to test through simulation since its functionality entirely relies on the Ultrasonic PMOD. So instead we tested this directly on hardware itself.

We initially created multiple slave registers to connect debug signals to. We then wrote test software for the Microblaze to continuously poll both the Obstacle Detector IP and Ultrasonic PMODs to check if they are consistent. Finally, some of these slave registers were removed, and only the essential ones were kept.

We also had issues with the Master interface reading garbage values from the Ultrasonic PMOD slaves. We used ILAs on the AXI interface ports to monitor and ensure that the protocol was being respected.

4.2. Vivado IP Library Blocks

The following major IP blocks were borrowed from Vivado's IP library.

4.2.1. Microblaze processor

IP name: Microblaze Ver 10.0

Processor to execute motor control logic based on LED position and obstacles.

4.2.2. AXI Stream Divider Generator

IP name: Divider Generator Ver 5.1

Configurable Divider IP with AXI Stream Interface.

4.2.3. AXI Stream FIFO

IP name: AXI4-Stream Data FIFO Ver 1.1

Data FIFO with AXI Stream interface.

4.2.4. AXI Lite

IP name: AXI Interconnect Ver 2.1

Bus for the processor to interface with motor drivers and custom IP block.

4.3. PMOD IP Blocks

The following are the PMOD IP blocks used to interface with the PMOD hardware.

4.3.2. PWM IP

IP name: PWM_v2.0 Ver. 2.0
Source: Digilent PMOD IP library [1]

IP block to generate a configurable PWM signal.

4.3.3. MAXSONAR PMOD IP

IP name: PmodMAXSONAR_v1_0 Ver. 1.0
Source: Modified from Digilent PMOD IP library [1]

IP block to interface with the Ultrasonic Range Detector PMOD and read distance values. The interface for this IP block initially used up an entire PMOD connector block consisting of 12 I/O pins, though it only used 6 of them. It was modified to use only 6 I/O pins. This allowed us to plug two Ultrasonic PMODs into one PMOD connector block using a cable splitter.

4.4. IP Blocks from Past Projects

4.4.1 video_in_ip

Source: "Drag and Stamp System" project page on GitHub [2]

This IP block was taken from the previous project called "Drag and Stamp". This IP connects the pixel data, vsync, and hsync signals from the camera PMOD to the Xilinx Video to AXI-4 Stream IP allowing us to use the camera pixel data without any additional configuration. To test that this IP was working as expected we used the AXI TFT controller, and AXI VDMA blocks to continuously display the camera image on a monitor through the VGA port.

4.5. Software Components

The following are the software components used in our design. They are all custom written, unless stated otherwise. All software runs on the Microblaze Processor.

The software runs in a continuous loop which reads data from the various sensors on the robot and uses it to control how the motors are driven. A different routine will be run and updated on each iteration depending on which of the three states the robot is currently in (AVOID, FOLLOW, SEARCH).

The three states/routines are described below.

4.5.1. Obstacle Avoidance (AVOID state)

Obstacle Detector IP block (described in Section 4.1.2) raises an interrupt if obstacles are detected in the vicinity of the robot. The software consists of an interrupt system that gets set up when the processor boots. The Obstacle Detector IP block is registered as an interrupt source with the interrupt handler. Upon receiving an interrupt, the interrupt handler will update the state to the AVOID state.

While in the avoid state the robot stops driving forward. The software polls the distances to obstacle from both the left and right sensors. If the left distance is shorter than the right the motor speeds are updated to rotate the robot to the right. If the right distance is shorter than the left the robot is rotated to the left. As the robot steers away from the obstacle, eventually both distances become greater than the minimum threshold distance, after which the robot transitions to the FOLLOW state.

Source of interrupt handler routine: Xilinx Interrupt Controller example [3]

4.5.2. Destination Follow (FOLLOW state)

The follow routine consumes the destination marker positions from the Light Detector IP (Through the AXI Stream Divider hooked up to the Microblaze Stream Interface). The value corresponds to the x coordinate of the pixel position of the target visible on the camera. This value is mapped to [-1.0, 1.0] with 0 signifying the center of the screen. This value is used to control the motors. If the object is on the left side the right motors are set to max power and the right motors speed are linearly interpolated between the minimum motor speed and the maximum motor speed depending on how far from the center the target is. The opposite happens if the target is on the left.

If the target is lost (x pixel value of 0) we transition to the destination search state.

4.5.3. Destination Search (SEARCH state)

The destination search routine runs when an LED is not visible. It performs the following steps:

(i) First rotates the robot approximately 360°.
(ii) Performs another random rotation between 0° and 360°.
(iii) It will then drive forwards for a set amount of time.
(iv) GOTO (i)

During this routine the robot will continuously consume target positions from the Light Detector. If the destination is found (non-zero x-position) during any point in this routine we transition to the Destination Follow state.

5. Description of Design Tree

The following is a brief description of the key files and folders in our project

- **src:** Contains all project files
    - **light_detector_system:** Project folder
        - **light_detector_system.xpr:** The project file used for this design
        - **light_detector_system.sdk**
            - **light_detector/src**
                - **helloworld.c:** Contains the main loop for the robot, sets up hardware and driving routines
                - **motor_controller.h:** Contains functions for initialization, enabling and disabling of motors. Also contains functions to set the direction, and testing the motors
                - **range_detector.h:** Contains the functions used to initialize the ultrasonic sensors and the range detector ip also sets up interrupts for the range detector ip and contains a function to poll the ultrasonic sensors for debugging
    - **light_detector_ip:** Contains the light detector ip files needed by IP manager
    - **range_detector_ip:** Contains all the range detector ip files needed by IP manager
    - **video_in_ip:** Contains all video in ip files needed by IP manager
- **doc:** Contains our final report, and final demo presentation slides

6. Tips and Tricks

Use the AXI Stream interface over AXI Lite whenever possible as it is much simpler than AXI Lite. For example we were originally considering using double buffering for our light detector so that it would read frames from memory, but it is much simpler to read pixel by pixel over AXI stream. It is also faster since there is no need to go into memory.

Connect IP block parameters to slave registers. This way changing the parameters can be done by Microblaze processor through software, and doesn't require recompiling the hardware design.

Leave important ILAs hooked in the design. This can allow for a quick diagnosis of failure symptoms, especially when integrating the design and testing on hardware.

References

[1] https://github.com/Digilent/vivado-library

[2] https://github.com/mankaijon/G1_DragandStampSystem

[3] https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/intc/examples/xintc_example.c