ECE532 Final Project Group Report

Self Driving Miniature Car

Team 7

Prepared by: Tony Kim, Evan Loe, Andrew Leung, Aiden Chow

April 7th, 2025

Table of Contents

1. Overview

1.1 Background and Motivation

The vehicle industry consists of a wide variety of engineering concepts and disciplinary fields, especially in the areas of electrical and computer engineering, and still proves to be an intriguing major engineering industry today. Specifically, self-driving vehicles represent a revolutionary step in transportation, combining advanced sensors, automation, and complex algorithms to create safer and more efficient roadways. We wish to explore, at a much smaller scale, some of these technologies to demonstrate a combination of a variety of peripherals, sensors and actuators, and algorithms that are well applicable to digital systems design.

1.2 Project goal

The goal of the project is to develop a miniature-scaled vehicle capable of navigating between lines and/or following lines (representing a road) and avoiding obstacles. The approach will consist of DC motor control for the miniature propulsion system, live camera interfaces and video displays, various sensors and actuators, and control and processing algorithms implemented in hardware and software to achieve the desired outcome.

1.3 System Level Overview

The project entails the design and build of a miniature self-driving car, where the Nexys Video Board is used to interact with a variety of sensors and actuators to achieve autonomous motion. For direction control, a camera is utilized and processed using HDMI input and downstream image processing to detect lines and compute the required angle for steering. This camera feed is also displayed live onto an external display via HDMI output demonstrating the vehicle's algorithms and logging useful data. Mechanically, a servo motor with a respective proportional-integral (PI) closed loop controller drives the steering and propulsion of the car is achieved with a DC motor controlled by an H-bridge controller. Additionally, the car is equipped with an ultrasonic range finder for the detection of obstacles in its path such that it can stop before impact. The vehicle itself is powered by a separate battery where a power monitor sensor is added to keep track of the battery's voltage level as well as the power consumption of the system. Software current protection is also added to keep the controllers, mainly the H-bridge controller, from exceeding its rated power limits.

The system block diagram can be seen in Figure 1. The microblaze soft processor is used along numerous existing and custom IPs.

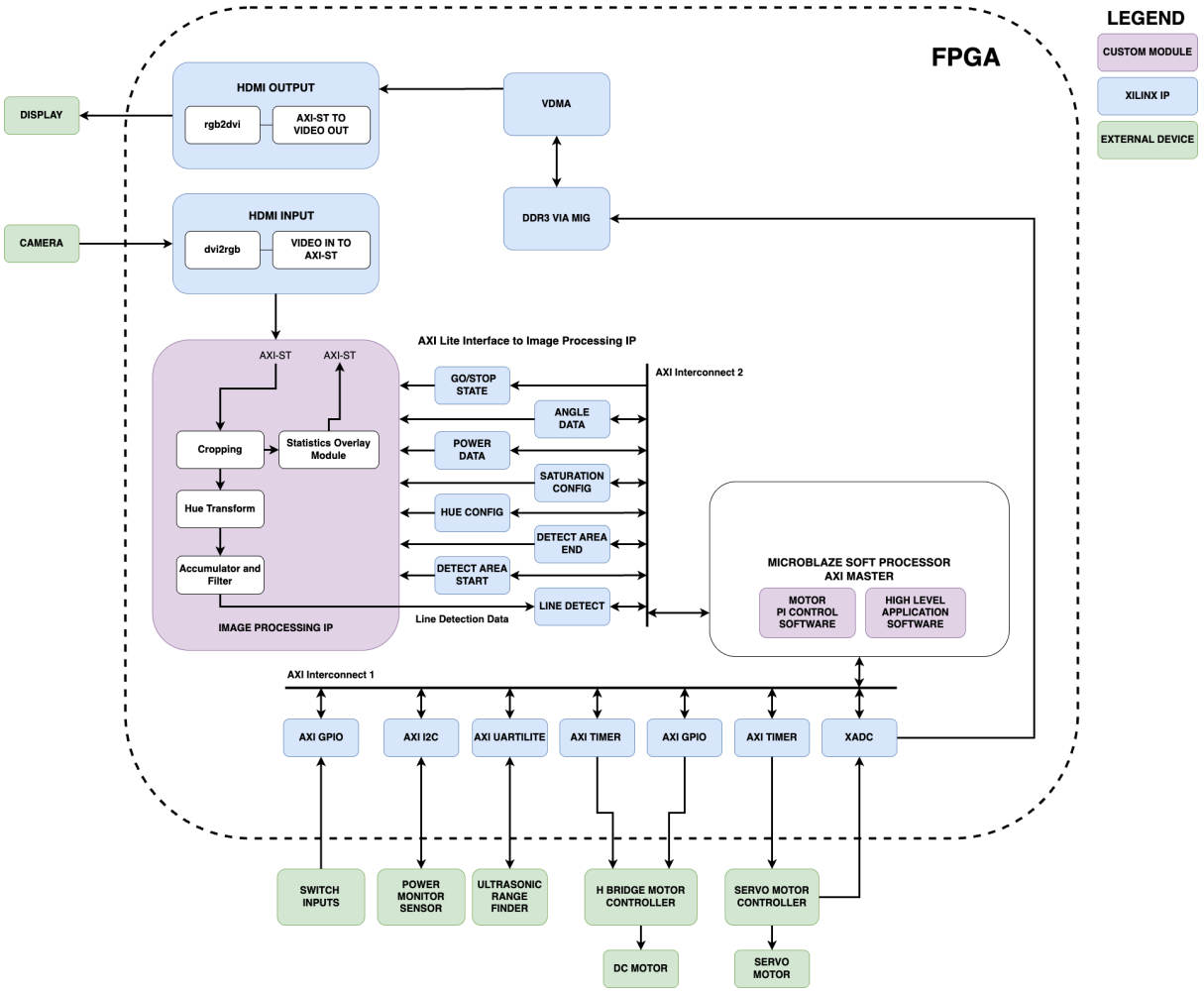

Figure 1. System block diagram

1.4 IP Overview

The project utilized many IPs to interface with the variety of sensors and actuators involved in the project. Custom IPs were designed for the image processing pipeline while all other IPs were sourced from Xilinx through the Vivado software. The table below gives an overview of the IPs used.

Table 1: IP Overview

IP	Description	Source
Microblaze	Xilinx's soft processor for initialising and controlling the various PMODs	Xilinx
AXI GPIO (x9)	Used for general-purpose I/O including motor direction control, overlay interface, and configuration. Note that AXI Lite Interface blocks to the Image Processing IP are essentially AXI GPIO IPs	Xilinx

AXI IIC	I2C interface used to communicate with the power monitor PMOD	Xilinx
AXI Timer (x2)	Used for generating PWM signals for DC motor via the H-bridge controller and servo control	Xilinx
AXI UARTLite	UART interface for communicating with the ultrasonic range finder	Xilinx
XADC	Used to read analog feedback from the servo motor for steering closed loop control	Xilinx
MIG	Generates a memory controller to interface with DDR memory (e.g., DDR3/DDR4) and provides the clocking infrastructure for the design.	Xilinx
RGB2DVI	Converts RGB video signals to TMDS format suitable for DVI/HDMI output.	Xilinx
DVI2RGB	Converts TMDS DVI/HDMI input signals to parallel RGB video signals	Xilinx
AXI-ST to Video Out	Converts AXI-Stream video data to parallel video output (video timing signals + RGB).	Xilinx
Video In to AXI-ST	Converts parallel video input into AXI-Stream format for processing within the FPGA.	Xilinx
Cropping	Crops the incoming video frame to a region of interest for the image processing pipeline	Custom
RGB to HSV	Convert RGB video data to HSV for more robust color detection	Custom
Edge detection	Detects the edge of the red line in the cropped video frame	Custom
Low Pass Filter	Smooths out the detected edge position using exponential averaging	Custom
Display Overlay	Overlays system information (e.g., voltage, current, angle, rangefinder status) on top of the HDMI output	Custom

2. Outcome

The outcome of the project successfully met the goals of developing a line-tracking self-driving miniature car. The car's performance was demonstrated by following a path marked with red electrical tape on the ground (see figure 2), which it was able to track and steer along accurately. Throughout operation, the car displayed real-time power measurements (voltage and current) and the angle of steering on its onboard display on top of its live camera feed, providing useful feedback on system behavior. Additionally, as it was equipped with an ultrasonic rangefinder, obstacles were placed at random in the path of the car, where the car was able to successfully detect the obstacle and stop before automatically moving forward when the obstacle was removed. Furthermore, to evaluate the car's responsiveness under different conditions, various speeds were tested by adjusting the duty cycle of the H-bridge motor control. Ultimately, the car met all the high level expectations of the project.

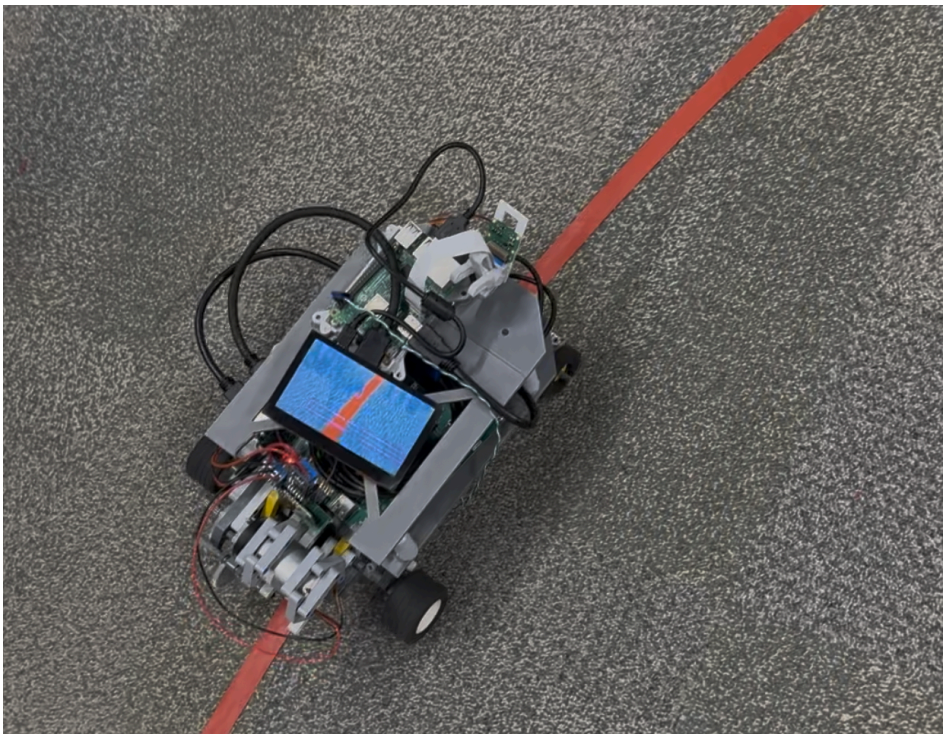

Figure 2. Snapshot of car during final demonstration and performance testing

2.1 Final Features Versus Original Proposed Features

During the development of the project, there were some adjustments made to the original proposed design for simplification and reduction of development time. Initially, the team wanted to implement Canny edge detection and Hough transform algorithms for line tracking. However, there were many challenges that were faced in the bring up of these algorithms, specifically with the integration of the Canny algorithm. An IP for this algorithm was sourced from a third party source where it was discovered that the sourced IP did not have a proper interface that could connect into our design. Thus, we spent time creating wrappers (AXI stream) in an attempt to utilize the source, but found ourselves spending days

debugging issues. Due to the limited time for the implementation of this project, we simplified our design to use colour detection to ensure our project was able to meet the initial goals of building a self-driving car. More details of the final design can be found in section 4 of this report.

The proposed design also featured additional peripherals including an accelerometer and a horn using the Nexys Video Board's audio output, but these were dropped due to time constraints, where the completion of the project with respect to the goals we set out in the beginning was prioritised. These peripherals were merely "nice-to-have" features such that they were not required.

2.2 Improvements and Next Steps

Several improvements and future steps can enhance the functionality of the car as well as further represent real-life self-driving vehicles.

In terms of the algorithms implemented on the current design, a more sophisticated algorithm for lane detection could have been used such as the original implementation plan of using Canny and Hough transform algorithms or utilizing CNN. Additionally, the steering control was implemented in software running on Microblaze using a proportional-integral (PI) controller, where the control loop could have been made more robust and faster by implementing it in hardware. Speed control is also something that could be improved, where the current design ran in open-loop control, requiring the user to set a specific speed using switch inputs. A closed loop controller could have been implemented (by adding an encoder to the DC motor for feedback) for the car to dynamically adjust its speed based on driving conditions - i.e. accelerating on straight paths, slowing down during turns, or reducing speed when obstacles are detected within a certain distance.

To further emulate real-world vehicles, more peripherals could be integrated such as headlights, blinkers, and a horn for honking. The performance of the current car relies that the lines are visible from the camera's perspective in which adding headlights would allow the car to navigate on its own dark environments. Additionally, remote FPGA programming would significantly improve development efficiency by eliminating the need to manually connect a programming cable for every update, while also reflecting how modern electric vehicles are managed today. Lastly, adding voice command functionality would create a more interactive system, allowing users to direct the car to specific locations using natural speech.

If the project were to be started over from the beginning, more efforts would have been put into the image processing component of the project such that more real-world applicable algorithms could have been implemented within the timeline of the project. Additionally, if someone were to take on the project from this point, the team would recommend implementing the improvements and additional features mentioned above in any order desired.

3. Project Schedule

3.1 Planned vs. Actual Work

The table below shows our original milestones of our project compared to the actual work we have done each week.

Table 2: Project Work Comparison

Planned Work	**Actual Work**
Milestone 1	
Focus on project setup and definition, ensuring all architectures, modules, and designs are ready to begin implementation. This includes: ● Project setup with Microblaze and peripherals blocks in Vivado ● Establish system bus and module architecture ● Finalize image processing pipeline, including algorithms that need to be implemented in hardware	● Searched for existing solutions and IPs to implement the Sobel and Hough Transform algorithms in hardware ● Setup HDMI passthrough example ● Researched on Hough transform in helping to determine the lines of an image ● Researched strategies to implement software algorithms in Vivado toolchains
Milestone 2	
Demonstrate high priority individual components of the project that are less complex. This includes: ● Demonstrate HDMI output working by writing to an external display ● Demonstrate spinning motors using H-bridge controller and servo motor controller ● Demonstrate ultrasonic rangefinder sensor readings of distances with tested accuracy with Nexys Video Board	● Debug an existing GitHub Repo to create a Vivado Project for Canny Edge Detection ● Research on using Vitis HLS for Hough Transform ● Started on custom RTL implementation of Hough Transform ● Developed and tested motor control systems (H-bridge, servo) using PWM and GPIO; verified using oscilloscope. ● Integrated UART interface with a rangefinder to display sensor data over serial communication.
Milestone 3	
Subsystem bring-up and more complex individual components such as video processing. This includes:	● Debug and enhance the third-party canny edge detector

• Demonstrate HDMI input working with HDMI output (perception subsystem) • Demonstrate accelerometer readings as well as its calibration with Nexys Video Board • Demonstrate implemented video processing hardware module and its testbenches	• Implementing the Hough Transform custom IP for parameterization and accumulate module • Developed I2C interface with power monitor module, reading voltage, current (verified with multimeter) • Developed ADC on the FPGA for feedback from servo motor (analog signal) • Developed software (partially tuned) PI controller (position control) for servo motor - able to take angle as an input and output a PWM duty cycle to control the motor
Milestone 4	
Subsystem bring-up and lower priority individual components of the project. This includes: • Demonstrate video processing algorithm correct behaviour on HDMI output (via display) • Demonstrate motor closed loop control (PID controller) • Demonstrate power monitor sensor correctly reading power measurements with Nexys Video Board	• Parametrized the canny edge detection module • Worked on the canny edge detection module into the pipeline • Validated hough transform parameterization module with developed testbench with AXI Stream VIP. Verified calculations of fixed point multiplication, sine and cosine tables, etc. • Integrated hough transform parameterization module with accumulator module and verified functionality through simulation
Milestone 5	
Demonstrate initial stages of the project's integration and the beginning of the car's mechanical assembly. This includes: • Demonstrate all subsystems working simultaneously on bench (i.e. motor turning based on perception data from camera input, correct data logging, etc.) • Demonstrate initial plans for car assembly	• Investigated voting algorithm implementation; planning integration with MicroBlaze • Debugged Canny Edge Detector and its testbench; identified issues with the ready/valid signal. • Developed AXI-Stream wrapper and FSM for padding to match expected frame size in Canny IP • Faced issues with Vivado IP packaging and automatic updates. • Assembled all car components; vehicle can now drive and turn on the ground.

Milestone 6	
Demonstrating a further tuned fully integrated system. This includes: ● Demonstrate the fully assembled car and its working functionalities (i.e. the car is able to drive forwards, follow lines, and avoid obstacles)	● Created RGB line tracking ip to replace hough transform ● Integrated RGB tracking with steering control ● Assembled components for car - made a power converter that outputs 5V for powering the camera ● Added current protection to design using the power monitor module readings
Buffer Week (leading up to final demonstration)	
No milestones planned for this week.	● Debug and testing of the RGB tracking module ● Full integration testing with all the components ● Implemented the display overlay module

3.2 Discussion

Throughout the development process, the hardware subsystems, such as the motor drivers, servo feedback, and power monitor, progressed smoothly. We were able to stay ahead of our schedule and assemble all the hardware components on time. Early verification using measurement tools like oscilloscopes and multimeters contributed to the stability and reliability of these hardware components.

On the other hand, we were behind schedule for the image processing pipeline developments. Multiple weeks were spent on understanding and developing the custom IPs for hough transform. In addition, we encountered a lot of issues with the third party IP for canny edge detection, particularly with its compatibility with sending AXI-ST data to downstream modules. Eventually, we shifted to RGB line tracking to meet our time constraint. This underscores the complexity of video processing IP, as well as the risks associated with using third party IP.

To address these challenges in future projects, more time should be explicitly allocated for thorough testing, debugging, and validation of video processing IPs. Additionally, integration efforts should begin earlier in the development cycle to quickly identify and resolve compatibility issues with third party IPs.

4. Detailed Final Design (IP Blocks and Systems)

The final design integrates various sensors and actuators into a custom built miniature scaled vehicle made with 3D printed components and lego parts. The figure below shows the fully assembled vehicle.

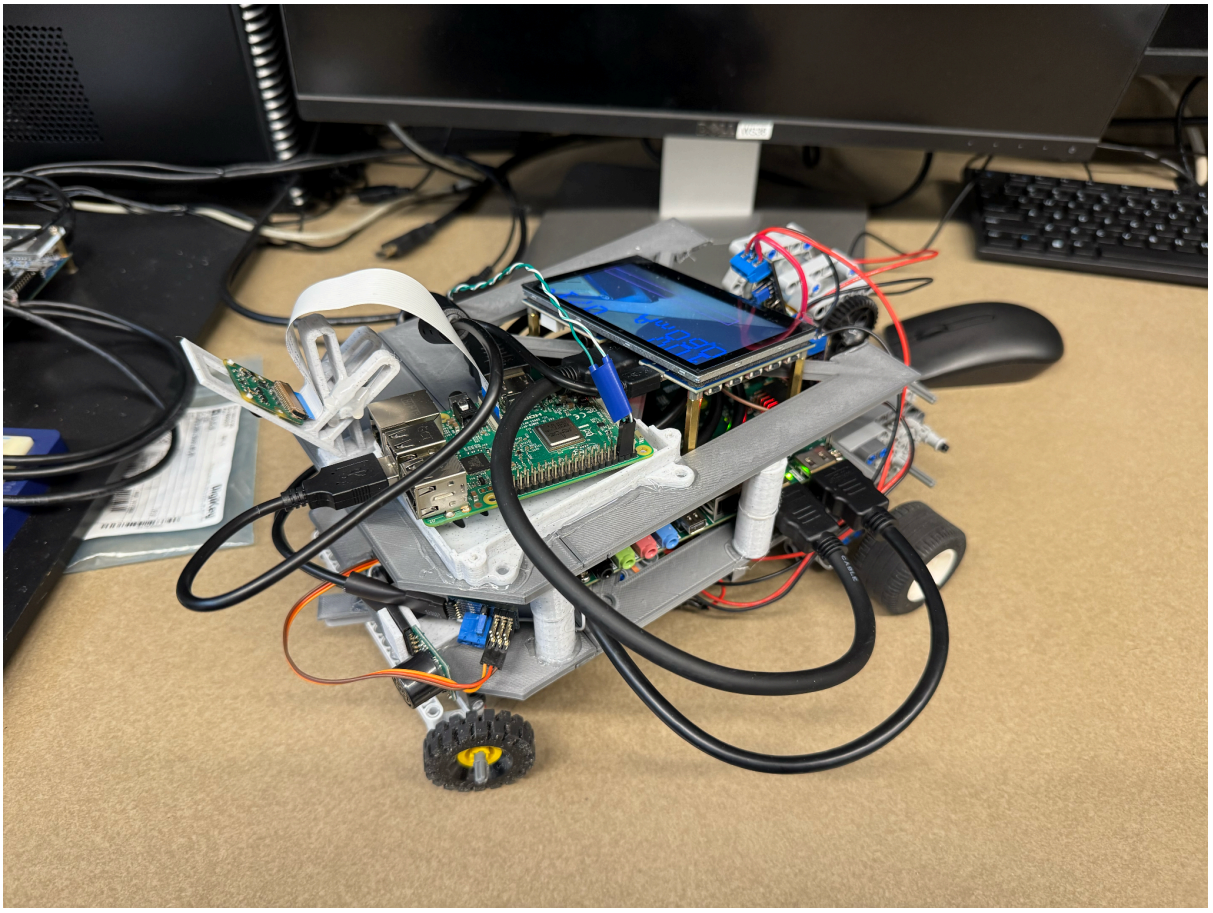

Figure 3. Assembled self driving vehicle

The Nexys Video Board is the main processing unit of the project. On the FPGA, numerous IPs have been designed and configured to interact with each of the sensors and actuators that build up the functionalities of the car.

4.1 Power Distribution and Wiring

The self-driving miniature car is battery powered by three 18650 lithium ion cells (LG MJ1) put in series, reaching approximately 12V to power the DC motor and the Nexys Video Board. However, as the design incorporates an external camera [1] using a raspberry PI and an external display which operates at 5V, multiple 12V to 5V step down converters [2] in parallel are used in the design. The figure below is the detailed wiring diagram of the entire system.

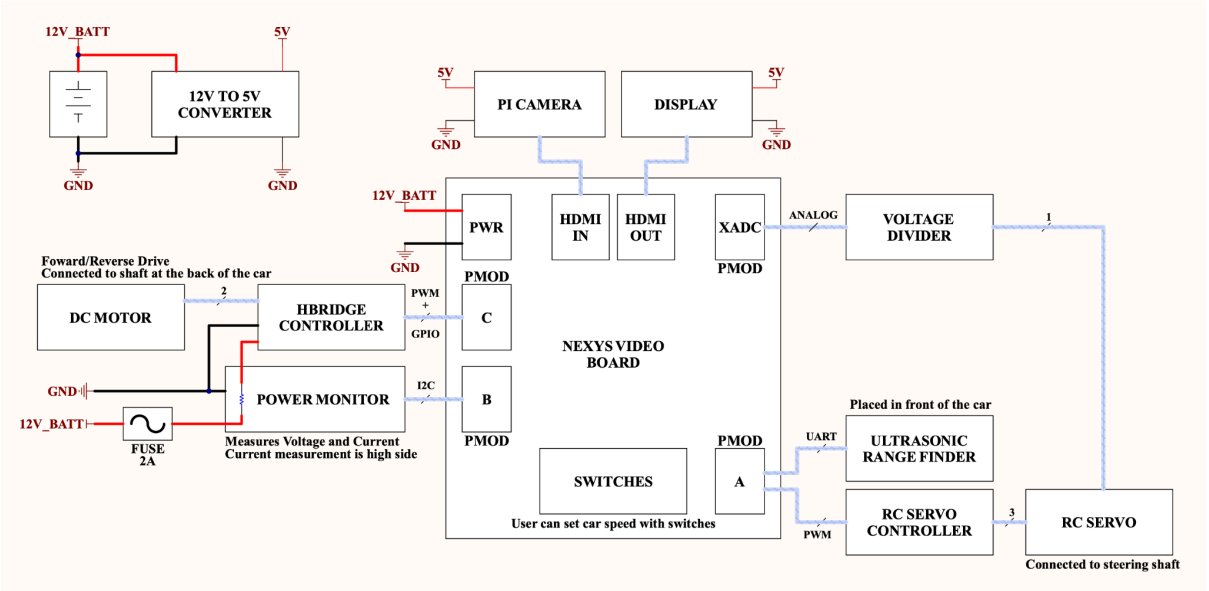

Figure 4. Self driving car wiring diagram

4.2 Propulsion System

The propulsion system of the vehicle is achieved with a 12V DC motor, where the HB3 H-bridge controller PMOD [3] is used to regulate the speed of the motor using pulse-width modulated (PWM) signals and the rotation direction with a single GPIO. The AXI Timer IP from the Xilinx library is used to control the PWM output signal where the PWM signal is mapped to PMOD C of the Nexys Video Board that is routed to the enable pin of the H-bridge controller. The AXI GPIO IP from the Xilinx library is configured to output mode for a singular GPIO that is routed to the direction pin of the H-bridge controller. These IPs are AXI slave modules in which Microblaze serves as the master, controlling duty cycle and frequency of the PWM and the assertion state of the GPIO. See section 4.7 Software on Microblaze for more details on the software control.

The motor is mounted onto the back of the vehicle and numerous gears are mechanically implemented to address the torque limitations that were empirically determined during testing. The figure below shows the mechanical mounting of the DC motor.

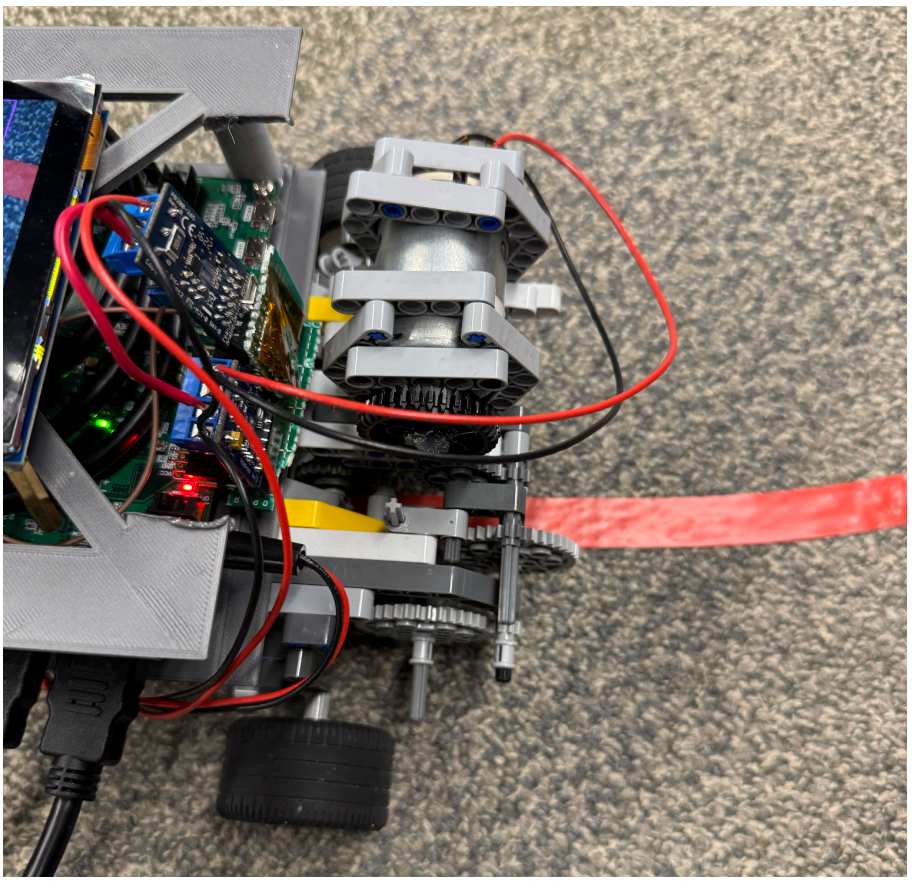

Figure 5. DC motor mechanical mounting on back of the car

4.3 Power Monitoring

Power monitoring consists of measuring voltage and current of the DC link between the battery and the DC motor using an I2C interface power monitor PMOD (PMON1) [4] . The Xilinx AXI IIC IP is configured to interact with the PMOD using standard I2C mode (100kHz), and external pull up resistors of 4.7kOhms are added on the SCL and SDA bus lines as the FPGA's internal pull up resistors were empirically found to be too weak. The AXI IIC IP is an AXI slave module to Microblaze, where the soft processor runs I2C protocol in polling mode, triggering voltage and current measurements with a software defined timer roughly every 10 Hz. The power monitor module is connected to PMOD B of the Nexys Video Board. See section 4.7 Software on Microblaze for more details on the software control.

4.4 Distance Detection

An ultrasonic range finder PMOD (MAXSONAR) [5] is placed at the front of the miniature vehicle to detect objects in the way of the driving path. This PMOD is interfaced with UART such that the AXI UARTLITE IP from Xilinx is utilized, configured at a baud rate of 9600 bps. The PMOD streams periodic measurements of distances where Microblaze, being the AXI Master of the AXI UARTLITE IP, reads distance measurements roughly every 00 Hz using polling mode. The ultrasonic range finder is routed to the top row of PMOD A of the

Nexys Video Board. See section 4.7 Software on Microblaze for more details on the software control. The figure below shows the front face of the vehicle where the ultrasonic range finder is found.

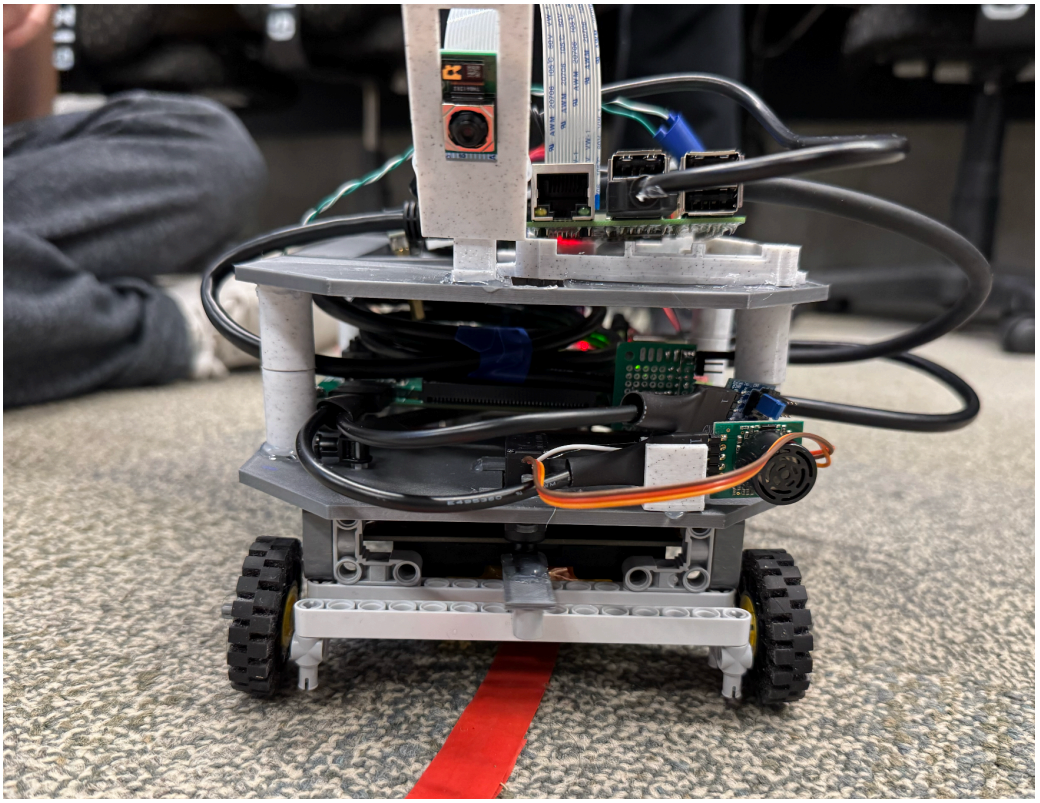

Figure 6. Front of the car. The ultrasonic range finder is mounted on the right side above the front tire

4.5 Steering Control

Steering is implemented with a singular RC servo motor [6] that comes with an analog feedback signal to indicate its position. For driving the servo motor, a servo controller PMOD [7] is used that takes a PWM signal as input. The Xilinx AXI Timer IP is used to generate the required PWM where the frequency of the PWM is configured to 50Hz and the PWM signal is routed to the top row of PMOD A to the corresponding pin on the servo controller. The feedback analog signal of the servo motor is inputted into the external auxiliary XADC channel using the XADC IP from Xilinx. The XADC block is configured in channel sequencer mode (although only 1 channel is actually used in the design), continuous timing mode, and a channel averaging of 16 samples is applied. Microblaze serves as the AXI Master for these two modules where in software, a PI controller is implemented to run closed loop position control for steering. The figure below shows the block diagram for the control algorithm that runs in Microblaze.

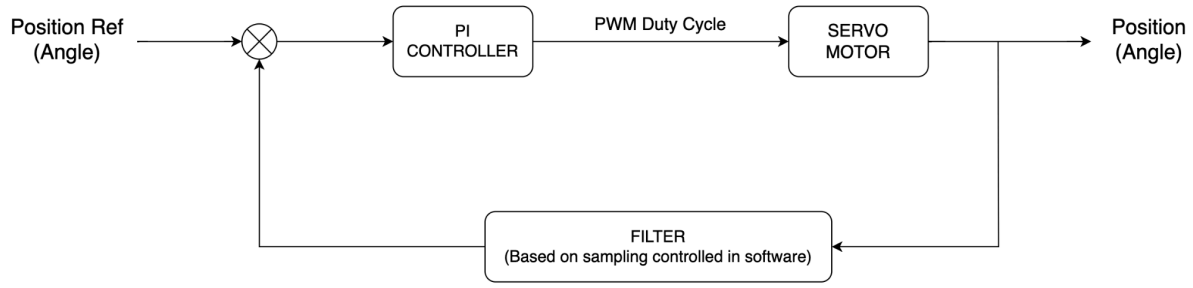

Figure 7. Steering position control loop block diagram

The PI controller is implemented in the Z-domain in discrete form to avoid complex computation that slows down the control loop frequency. The controller is modeled in the equations below, showing the proportional, integral, and the controller output.

$$p[n] = Kp * e[n]$$

$$i[n] = \frac{Ki * T}{2} * (e[n] - e[n-1]) + i[n-1]$$

$$u[n] = p[n] + i[n]$$

where Kp and Ki are the proportional and integral constants respectively, p[n] is the proportional term at step n, i[n] is the integral term at step n, u[n] is the control output at step n, T is the sample time, e[n] is the error at step n. Kp is set to 0.0035, Ki is set to 0.00505, and T is set to 0.01, which are all values tuned empirically.

In the software implementation, integration and saturation limits are added to avoid anti-windup and out of bound controls that may damage the servo motor. Thus, the servo motor is limited to the range from 30 to 150 degrees referencing the front flat face of the car. Furthermore, as the feedback signal is analog (0V to 2.5V) representing the position of the car, ADC calibration was conducted to determine the linear relationship between analog voltage and angle. The calibration results can be seen in the graph below, and the line of best fit is used in the software to compute the angle from the feedback signal which completes the closed loop controller. See section 4.7 Software on Microblaze for more details on the software control.

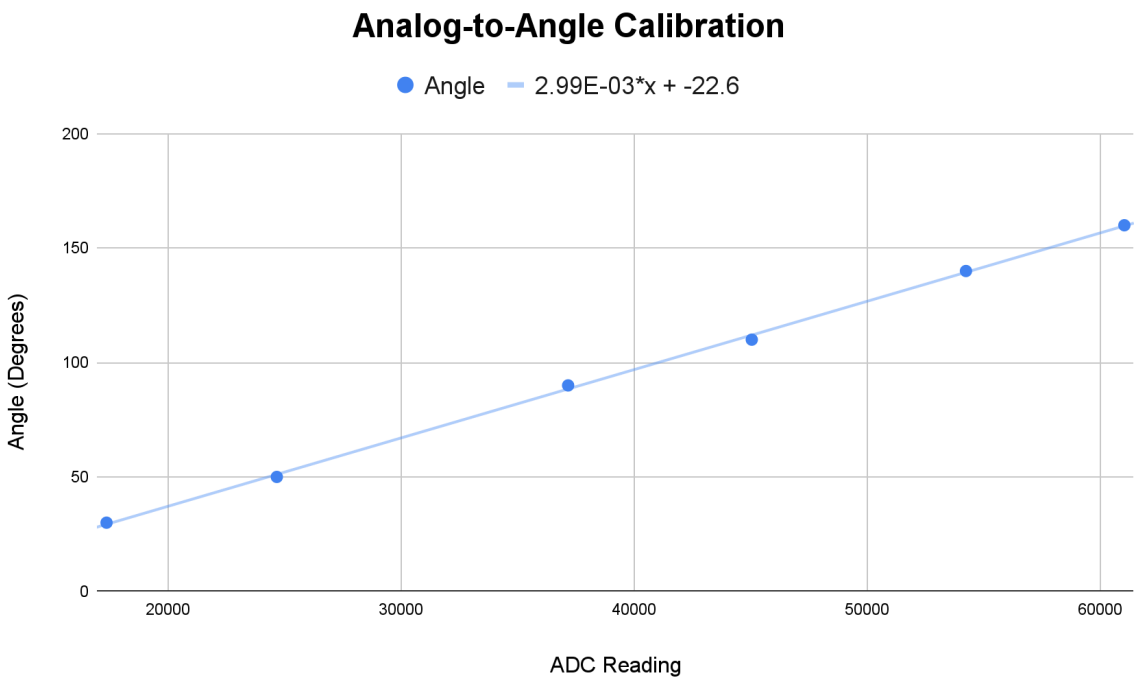

Figure 8. ADC reading to angle calibration results

Mechanically, the servo motor is attached to the steering shaft that enables a single motor to steer the entire car. The figure below shows the mechanical assembly of the steering shaft and servo motor attachment.

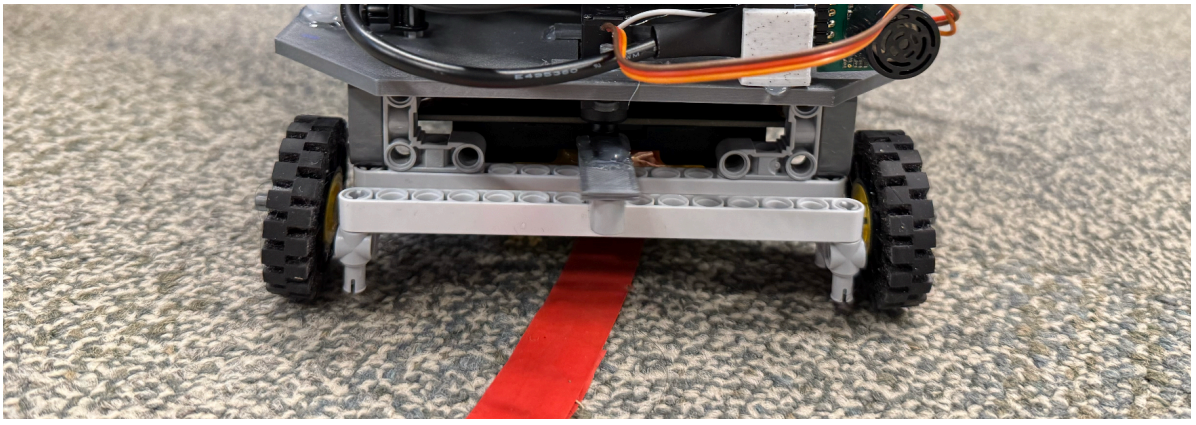

Figure 9. Steering shaft mechanically assembly

4.6 Image Processing Pipeline

The goal of the Image Processing Pipeline is to extract the position of the coloured line from the frame. If the line is far to the right, we know that our car is left of the line and so we should tell the wheels to turn to the right, and vice versa. Our method for detecting the position of the line relative to the frame involves cropping the frame and only looking at a small section. In this section, we keep track of when the colour hue suddenly changes to our desired hue, such as red as we used for our demo.

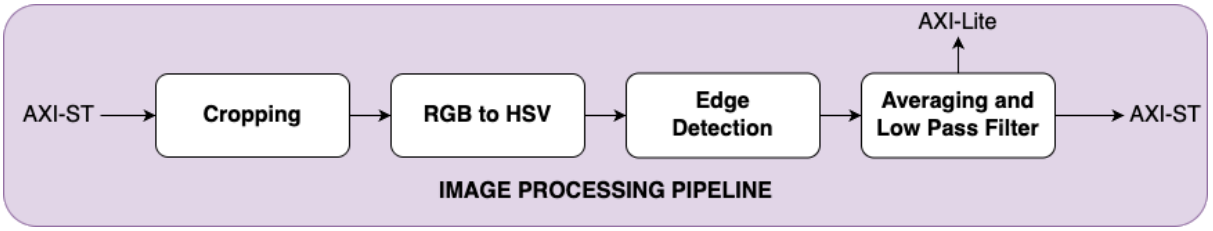

Figure 10. Image Processing Pipeline

The Image Processing Pipeline first gets its data from the HDMI input which is the unprocessed video from the Raspberry Pi Camera, where Vivado IPs DVI2RGB and Video-In to AXI-ST convert the incoming frame into an AXI-ST. Through microblaze, we can write to registers inside the IP to change the desired cropping area position and size. After the frame is cropped, we transform the pixel data which is in the RGB colour space to the HSV colour space.

Although RGB is the standard format for representing pixel data for displays, it is much less robust to try to detect a specific hue using this format such as yellow or red, as it is susceptible to differences in lighting. The HSV colour space represents colours as a hue from 0-360º (0º, 120º, 240º representing red, green, and blue, respectively), a saturation value from 0-100, and value (or brightness) value from 0-100.

To convert RGB to HSV, we use the following formulas:

$$max = max(R, G, B) \qquad min = min(R, G, B) \qquad \Delta = max - min$$

$$if\ C_{max} = Red,\ hue = 60^{o} \times ((G - B)/\Delta + 6\ if\ (G - B) < 0)$$

$$else\ hue = 0,\ saturation = 0,\ value = 0$$

The detection of the edge of the line is done by detecting the position at which the pixels in the cropped area suddenly change to the expected colour of the line. Registers in our IP customizable in software allow us to set thresholds for what hue, saturation, and values we accept as being the line colour. To set red as the expected hue for the line, we set our thresholds, such that

$$-thresh_{hue} < hue < thresh_{hue}$$
$$saturation > thresh_{sat}$$
$$value > thresh_{value}$$

We repeat the edge detection for every line in the frame and run it through an averaging and low pass filter with the form

$$edgestart_{avg} = 0.9 * edgestart_{avg} + 0.1 * edgestart_{new}$$

In the case that the edge is not detected, $edgestart_{avg}$ will not be updated and will hold its value. In practice, this means that the car will continue driving straight based on the last known line position.

The AXI-ST data is fully passed through our IP and transferred into external memory by the VDMA before it is read out of memory and sent to the HDMI output through the AXI-ST to the Video-Out and RGB2DVI ips.

The constantly updated value of the averaged position of the detected line is available in a register for Microblaze to read and tweak the servo accordingly.

4.7 Display Overlay

The display overlay module is a custom IP that overlays statistics on top of the HDMI output. It displays the voltage level and the amount of current supplied by the battery, status of the range finder (whether an obstacle is detected or not), as well as the turning angle of the car. This module does not add any extra latency to process the data.

It first gets the values to display from Microblaze through AXI Lite interfaces implemented with AXI GPIO IPs. The displayed value is separated into the hundredth, tenth, and unit digit. Then, it gets the AXI-ST data from the image processing pipeline and keeps track of the row and column of the pixel that the module is currently processing, using a row counter and a column counter.

Each character has a 128x128 pixel resolution. Pixels are grouped into 16x16 groups to display an 8x8 bitmap character. These numbers are exponents of 2 such that a bitmask can be used on the row and column of the pixel to determine which character and which group of pixels it belongs to. Thus, bits 0 to 3 of the counters are not used since they are all part of the 16x16 group. Bits 4 to 6 are used for determining the row and column of the bitmap within the character to be written, and the rest of the upper bits are used to determine the position of the character on the display.

A 15x8x8 ROM for representing the characters for the display, including 0-9, V, m, A, º, stop, and a blank character. Each address stores a 1 bit data representing if that position should be written. The ROM is indexed by the value's digit, and bits 6:4 of the row and column pixel counter.

Finally, it outputs the data through AXI-ST to the VDMA.

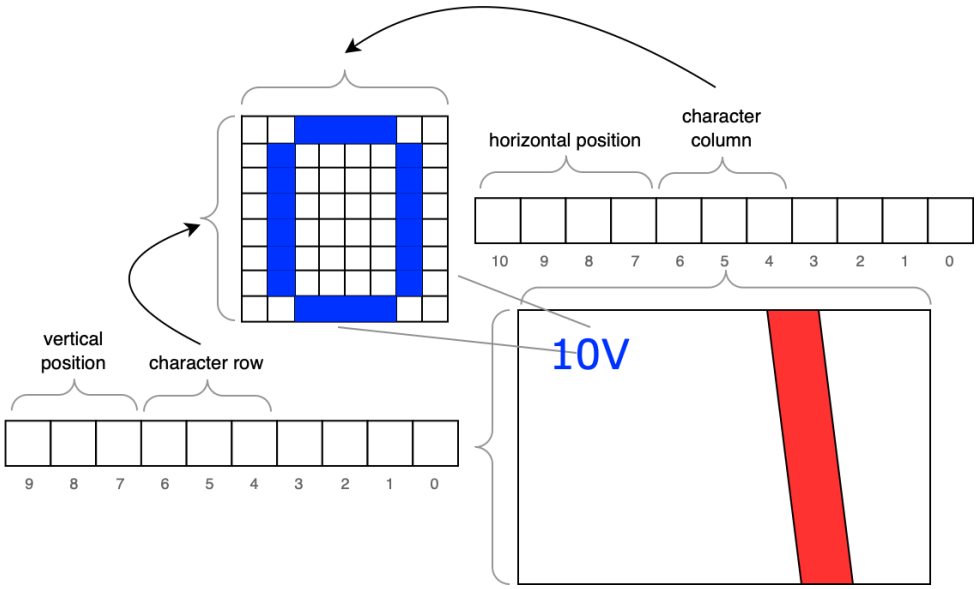

Figure 11. Character Display Logic Using Counters

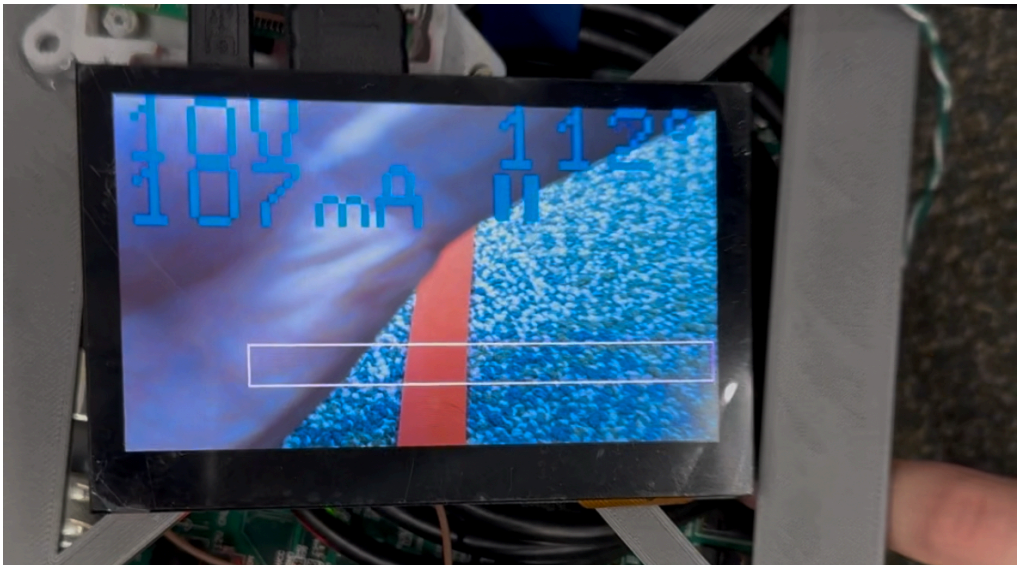

Figure 12. Display of the Car

4.8 Microblaze

4.8.1 Microblaze IP configuration

The Microblaze IP followed most of the default configurations when instantiated through Vivado. Most notably, the processor implementation was chosen to be 32 and a floating point unit was added to support computations done in software. Furthermore, local memory was increased to 64KB to support the size of the program.

4.8.2 Microblaze Embedded Software

The embedded software architecture consists of six software modules each responsible for their own peripheral handlers and corresponding variables. This design approach is used to keep the software modular allowing for easier debugging and enabling/disabling of features as needed.

Table 3. Software module descriptions

Software Module	Description
H Bridge Control	Controls PWM to regulate various speeds. Switches on the FPGA are used to set the duty cycle and the H-bridge will stop the motor if overcurrent is detected or an obstacle is in the path of the vehicle
Servo Motor Control	Runs the PI controller for the steering system of the vehicle. Inputs are the analog feedback signal and the reference angle command and the output is a PWM signal used to control the servo motor
Range Finder	Reads distance measurements from the ultrasonic range finder PMOD via UART
Power Monitor	Reads voltage and current from the power monitor PMOD via I2C
Line Tracing	Interfaces with the image processing custom IPs and outputs an angle that serves as the input to the steering controller
Display Overlay	Reads angle, voltage, and current from software and updates the HDMI display

Each software module includes its own initialization function as well as a periodic function that are both called in the main function of the code. The periodic functions are run in the event loop for continuous operation of the software where software timers are used to control the frequency at which each function is called. Figure 13 summarizes the software flow.

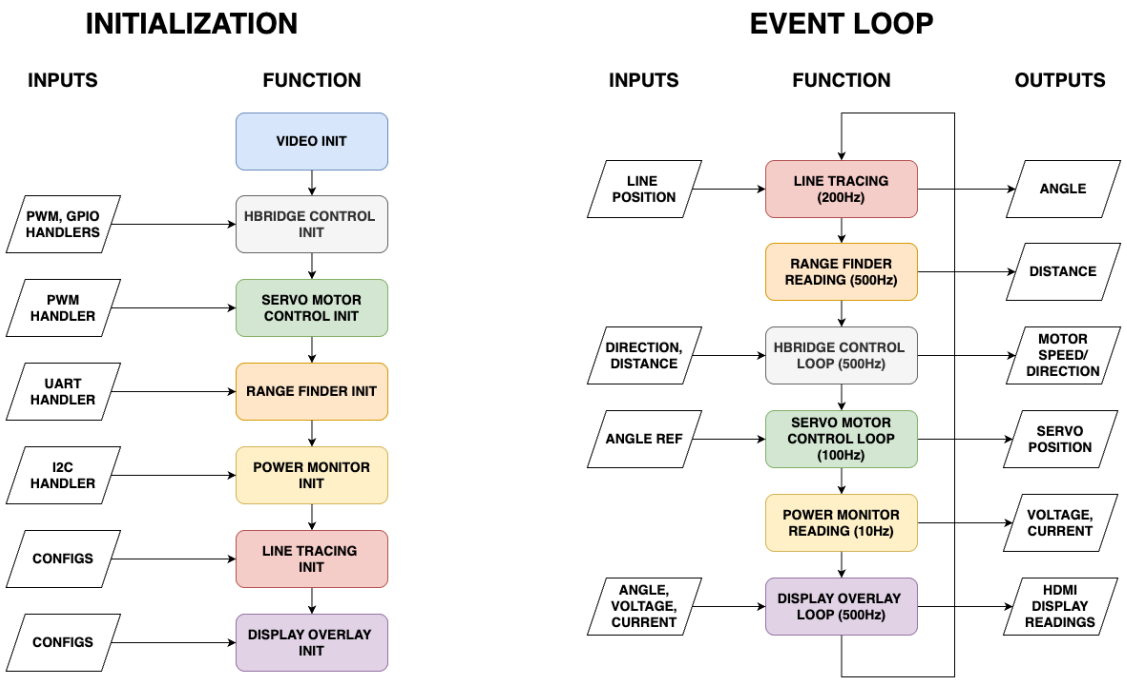

Figure 13. Software architecture and logic flow on Microblaze

5. Design Tree

Our project is accessible at https://github.com/tonykim07/ece532-self-driving-car. Along with our project source rtl, our project has been archived as a tcl file which can be opened using Vivado 2018.3. There is also a project archive in case there is trouble recreating the project with the tcl script.

The project is organized with subfolders as follows:

- **assets:** Images and misc for the README
- **docs:** PDF of this final report
- **hw:**
    - **hw/rtl**: Contains all the raw sources of our custom IPs
    - **hw/sanity.srcs:** Contains the source files required by the tcl script
    - **hw/self_driving_car.tcl:** The tcl script used to recreate the project
- **sw:** Contains the software source code (C code) for the Microblaze soft processor
- **ece532_self_driving_car_project_archive.zip:** Archive of our entire Vivado 2018.3 project

In addition to the subfolder organization, a README file is placed at the top level of the design tree that gives an overview. A video is also added showing a live demonstration of our project.

6. Tips and Tricks

The completion of this project entailed many challenges and valuable lessons that future students can gain advice from. Especially with a hands-on project, elements such as time management, proper validation procedures, and not being lazy are crucial towards a successful project.

One of the most important takeaways is the necessity of starting early and allocating enough time for integration. Throughout the project, we found that the most time was spent during the integration phases where designs and IPs do not seamlessly fit together as expected. As the semester progressed, we found ourselves in excellent shape with individual components brought up, but gradually saw delays in our timelines increased as integration of parts occurred. Many aspects of hardware design, such as the Vivado software, are quite prone to unexpected bugs, which can be quite stressful during busy moments in the project, so it is very important to account for delays in the project timeline. Additionally, always having a backup plan is essential. Especially for an ambitious project that needs to be done within a single semester, it is not unlikely that something does not work. Our project went through multiple simplification changes to ensure that the project can be fully presented at the final demonstration. It is more important to have a finished product even though it is easier than a complex project that is not done.

Future students should also be cautious about relying too much on third-party sources. While external IPs or reference designs may seem like convenient solutions, they often come with integration challenges. Many third-party IPs are not necessarily compatible with your custom designs, and attempting to adapt them can sometimes take more effort than developing a solution from scratch. Trusting in your own abilities and fully understanding your system's architecture will ultimately lead to a more reliable design that you are able to debug much quicker than trying to figure out someone else's design. It is highly encouraged to not become lazy by trying to find solutions online rather than implementing it on your own. If you are taking this course, you are more than capable of solving problems and implementing good digital designs.

Furthermore, proper validation should never be overlooked such that proper functionality of components or subsystems should not be assumed to work just because it passes a single test case. Writing thorough test benches and verifying corner cases is essential for ensuring reliability. Debugging during the integration phase is particularly difficult, so it is important to confirm that individual modules function correctly before combining them. The integration process is often the most challenging part of the project, and overlooking small errors in individual components can lead to compounded issues later on. By rigorously testing each part of the system before integration, you can avoid chasing down obscure bugs when the project is in its final stages. Ultimately, success in this project revolves around realistic planning, systematic debugging, and a proactive mindset toward problem-solving.

7. References

[1] PiShop.ca, "Raspberry Pi Camera Module 3," [Online]. Available: https://www.pishop.ca/product/raspberry-pi-camera-module-3/. [Accessed: 07-Apr-2025].

[2] DigiKey, "TMR 6-1211 Traco Power," [Online]. Available: https://www.digikey.ca/en/products/detail/traco-power/TMR-6-1211/9383427. [Accessed: 07-Apr-2025].

[3] Digilent, "Pmod HB3 Reference Manual," [Online]. Available: https://digilent.com/reference/pmod/pmodhb3/start. [Accessed: 07-Apr-2025].

[4] Digilent, "Pmod PMON1 Reference Manual," [Online]. Available: https://digilent.com/reference/pmod/pmodpmon1/start. [Accessed: 07-Apr-2025].

[5] Digilent, "Pmod MAXSONAR Reference Manual," [Online]. Available: https://digilent.com/reference/pmod/pmodmaxsonar/reference-manual. [Accessed: 07-Apr-2025].

[6] DigiKey, "1449 Adafruit Industries LLC," [Online]. Available: https://www.digikey.ca/en/products/detail/adafruit-industries-llc/1449/5154664. [Accessed: 07-Apr-2025].

[7] Digilent, "Pmod CON3 Reference Manual," [Online]. Available: https://digilent.com/reference/pmod/pmodcon3/start. [Accessed: 07-Apr-2025].