# ECE532 Final Report

## WiFi Assisted Autonomous Parallel Parking

Apr. 9 2018

Qiu Yuan Chen
Cheng Ling He
Jing Xie

# 1. Overview

## 1.1 Introduction

Today we live in an era where artificial intelligence (AI) is the up-rising trend in every modern industry. When we talk about AI, the first few things that come out of the top of our head are probably robotics, AlphaGo, autonomous vehicles, etc. Our team are particularly interested in autonomous vehicles, as self-driving capable cars like Tesla are gradually populating the roads and getting more people's attention. One of the important features of self-driving cars is that they do not only drive on its own but they are also able to park into an available spot autonomously without bumping into surrounding obstacles nor parking over the lines. To many people, this may seem to only exist in science fiction movies, but it is actually happening around us and is very likely to become more popular in the near future. Many people are struggling when they try to parallel-park their cars into a tight spot. How can the self-driving cars do it perfectly every time? It is this curiosity that motivates our team to explore into auto parking and build a project upon this idea.

Our project aims to reproduce this autonomous parking system on a model car that can autonomously perform a parallel parking into a designated spot. The system will be assisted by sensors and camera taking pictures for image processing to identify the available parking spot and potential obstacles in the surrounding environment. It will self-adjust when performing parking to avoid collisions and eventually park within the lines. On top of that, we are also bringing in the concept of IoT to our system. Prior to the parking procedure, it will wirelessly query the parking server with the present parking spot to validate the availability of the spot. This is to solve the issue when in reality the car can sometimes accidentally park into other people's private/reserved parking spots. It can also help the central server to manage the parking lot more efficiently.

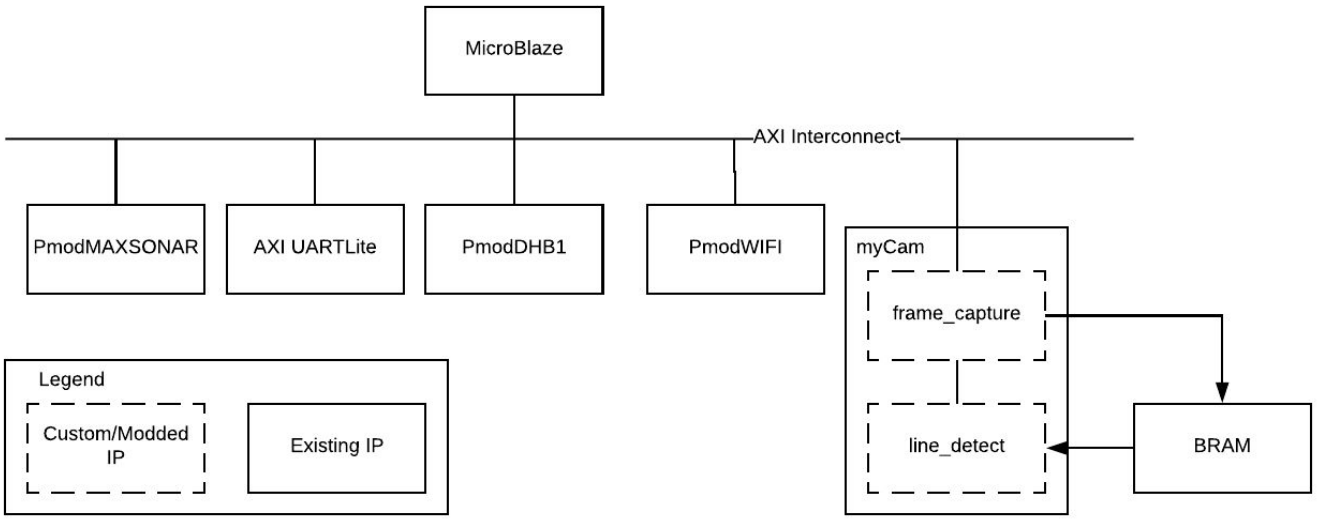
## 1.2 Project Goal:

The goal of our project is an model car which queries wirelessly the availability of a parking spot through a server and performs autonomous parallel parking into the spot if it is available.

## 1.3 Functional Requirement

1. Can detect parking location with suitable geometry
2. Can query the availability of a parking location over internet
3. Can parallel park into suitable parking location
4. Must stop the car to prevent potential collision

## 1.4 Block Diagram




From the top level, our car will have a system that consists of 5 major blocks:

1. A block that is responsible for controlling the physical car. This includes moving forward and backward, steering and stopping.
2. A block to control the ultrasonic sensors. These sensors will be used to find a suitable parking spot that is constrained by physical objects. I.e. another car. These sensors will also guard the car from potential collisions during any maneuvers.
3. A block that controls a camera. This will be used to find a suitable parking spot that is constrained by physical parking lines.
4. A block to handle the wireless internet connection, which is responsible for sending an identifier of a specific parking location to a parking server to inquire its availability.
5. A MicroBlaze processor that will oversee and control all the above blocks.

## 1.5 Brief description of IPs used

| IP name | Origin | Description |
|---|---|---|
| MicroBlaze | Xilinx | Soft processor |
| myCam | Modified from Piazza resource | Top level IP that incorporates the following two modules |
| frame_capture | Modified from Piazza resource | Single trigger capture from pmod camera to BRAM |
| line_detection | User | Analyze image saved in BRAM and check if a line exists |
| BRAM | Xilinx | Temporary storage for captured images |
| AXI UARTLite | Xilinx | UART interface |
| PmodMAXSONAR | Digilent | Controller for PMOD Ultrasonic distance sensor |
| PmodDHB1 | Digilent | Controller for PMOD Dual H-bridge |
| Pmod WiFi | Digilent | Controller for PMOD WiFi |

# 2. Outcome

## 2.1 Feature completion

| Requirement | Status |
|---|---|
| Can detect parking location with suitable geometry | Complete |
| Can query the availability of a parking location over internet | Partially complete |
| Can parallel park into suitable parking location | Complete |
| Must stop the car to prevent potential collision | Complete |

We have met 3 out of the 4 initial requirements for the project. The querying of availability of a parking location over WiFi was not completed fully. Instead of the parking server automatically replying the availability of the parking location after checking a database, there needs to be manual confirmation using an interactive Python prompt to start the parking process.We did this scale down to cut the work in the software side and to focus on optimizing the hardware line detection algorithm for better speed.

## 2.2 Future improvements

Currently, the auto parking is done by mimicking recommended parking maneuvers taught in driving schools. This type of hard-coding may lead to poor generalization to real world scenarios. A good improvements that could be made would be to let the car autonomously figure out the best actions and path to park the car.

## 2.3 Changes if restart from scratch

If we were to restart the project from scratch, we would try to build a better physical rigging structure from the beginning to mount the different Pmod components. We have found multiple times that the design was behaving wrongly due to sudden alterations of sensor placement caused by the unstable mounting.

We would also try to figure out ways to incorporate more Pmod ports to have more detection sensors all around the vehicle. Potentially we would replan the project using two nexys

board instead of one where one board will have a multitude of sensors and cameras while the other is connected to the dual H-bridge, WiFi module etc.

## 2.4 Recommended next steps

If a group will take over our project currently, the recommended next step is first to complete the unfinished confirmation over WiFi functionality. They should look into the MicroBlaze code and modify it to send a request over WiFi when a suitable parking location is detected. The python server on the PC also needs to be modified to reply with the availability status of the parking location after confirming with a local database.

# 3. Project Schedule

Refer to Appendix A and Appendix B for the original milestone proposal and actual weekly accomplishment.

| Milestone # | Status |
|---|---|
| 1 | On time |
| 2 | On time |
| 3 | Ahead |
| 4 | On time |
| 5 | Delayed |
| 6 | Delayed |
| 7 | On time |

The project mostly went as planned. The only fall back was for the integration of the different module for milestone 5 and 6. The milestone #6 suggested that the project should be completed and be in testing phase. However, we have mis-predicted the difficulty involved with the integration and so the project was completed and tested in week 7 instead.

# 4. Description of the Blocks

## 4.1 MicroBlaze

The MicroBlaze soft processor is directly instantiated from the IP catalog. Here are final address ranges of the memory mapping:

| | | | |
|---|---|---|---|
| PmodDHB1_0 | 0x00010000 | 0x00010fff | MOTOR_FB_AXI |
| myCam_0 | 0x44a40000 | 0x44a4ffff | S00_AXI |
| PmodDHB1_0 | 0x00012000 | 0x00013fff | AXI_LITE_GPIO |
| PmodWIFI_0 | 0x44a00000 | 0x44a0ffff | AXI_LITE_SPI |
| PmodWIFI_0 | 0x44a20000 | 0x44a2ffff | AXI_LITE_WFCS |
| microblaze_0_axi_intc | 0x41200000 | 0x4120ffff | s_axi |
| PmodWIFI_0 | 0x44a10000 | 0x44a1ffff | S_AXI_TIMER |
| PmodDHB1_0 | 0x00011000 | 0x00011fff | PWM_AXI |
| axi_uartlite_0 | 0x40600000 | 0x4060ffff | S_AXI |
| microblaze_0_local_memory_dlmb_bram_if_cntlr | 0x00000000 | 0x00007fff | SLMB |
| PmodMAXSONAR_0 | 0x00008000 | 0x00008fff | AXI_LITE_GPIO |
| PmodWIFI_0 | 0x44a30000 | 0x44a3ffff | AXI_LITE_WFG... |

## 4.2 BRAM

The BRAM is directly from the vivado IP catalog and is instantiated by the ov7670_top top module code included in the **532_pmod_camera_dist.zip** on Piazza. It has a capacity to store 76800(320 * 240) 16 bit(vg444) pixels.

## 4.3 PmodMAXSONAR

We directly used the IP code that came with vivado IP library under vivado-library-master\ip\Pmods\PmodMAXSONAR_v1_0. We include this IP to layout design and connect the physical sensor to one of the Nexys4 pmod port.

## 4.4 myCam (Custom IP)

The custom IP block “myCam” is implemented based on an AXI lite slave IP. It wraps the custom Pmod camera plus image processing module “ov7670_top” and extends signals that eventually get connected to the BRAM and pins on the board on the top level. The custom module interacts with the processor by reading and writing values to the slave registers provided in the AXI lite slave. Within the custom module, it instantiates the frame_capture and line_detection sub-modules. It also maintains a simple FSM logic that will change the image capture mode based on written values to the slave registers by the processor and coordinate between frame_capture and line_detection.

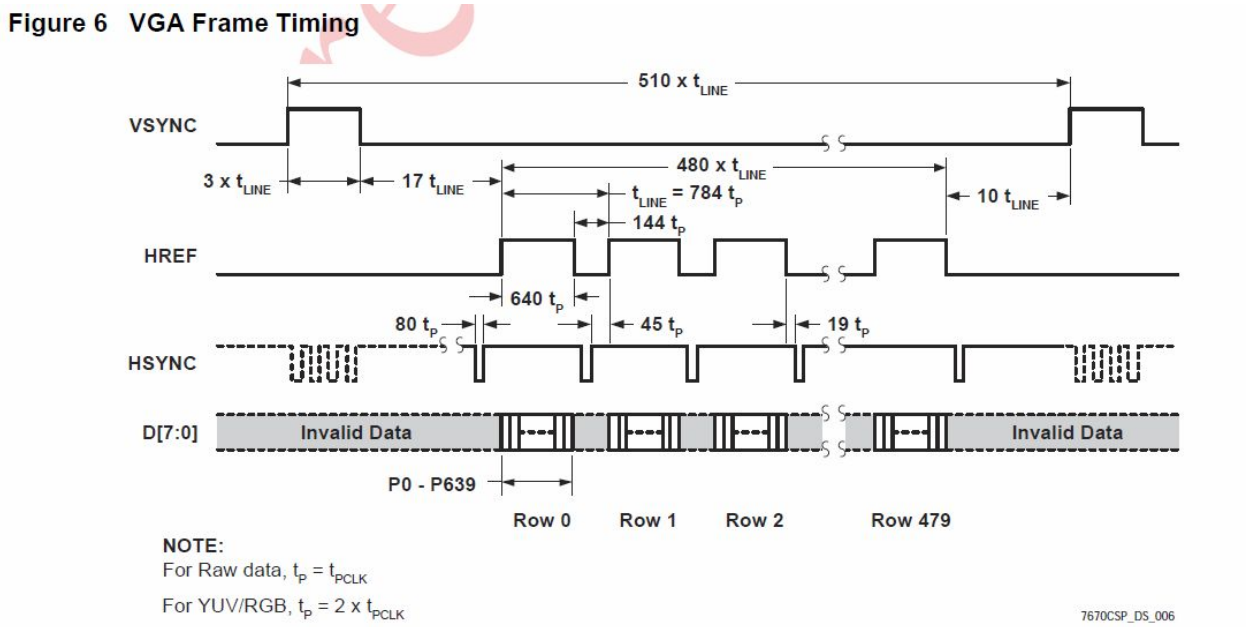
| Slave register # | R/W by module | Usage |
|---|---|---|
| 0 | R | Image capture command |
| 1 | R | Video stream command |
| 2 | W | Line detection algorithm done signal |
| 3 | W | Line detection result |

AXI Slave Registers Descriptions

### 4.4.1 frame_capture

The ov7670_capture module from **532_pmod_camera_dist.zip** originally featured the continuous capture of frames like a video camera. The code was modified so that it will wait for a trigger signal to begin the capture of a frame, and raise a completion signal when the whole frame is captured. This change was made because our project does not require line detection at all times but instead at specific moments on demand.

It is not feasible to directly trigger the camera using a signal because the camera is triggered internally by the VSYNC signal from the on-chip timing generator. Instead, we decided to control the write-enable signal for the BRAM that acts like the image buffer. To capture a single image on trigger, we turn off write-enable by default, and turns it on at the positive edge of the VSYNC signal when the trigger is high. Every time the VSYNC signal transitions from low to high, it marks the beginning of a new frame, and a counter clocked by the camera timing generator will begin counting the pixels. Once the pixel limit is reached, a done signal will be raised, and the write-enabled can be turned back low again.




Ov7670 camera VGA timing diagram from datasheet

## 4.4.2 line_detection

After a full image frame is captured and stored in BRAM, the upper module will change the state of the control FSM so that it temporarily disables writing of a new image to the frame buffer and gives the access control to line_detection module. The line_detection module then starts reading the image from the frame buffer pixel by pixel and processes each pixel given its relative pixel location on the image.

It first converts each pixel from RGB444 to grayscale for the ease of processing, and it then classifies the pixel to completely white or black based on a predetermined threshold. In this way, it eliminates most of the noises from the image and the image will be mostly left with a potential blackline of interest and the white background.

```
// Y = 0.25R + 0.5G + 0.25B;
assign grayscale = (frame_pixel[11:8]>>2) + (frame_pixel[7:4]>>1) +
(frame_pixel[3:0]>>2);
assign blackwhite = grayscale <= threshold ? 0 : 4'b1111;
```

After we obtain the image with minimum noises, we set up a thin rectangle bounding box in the center of the image where the blackline is supposed to be in order for the line to be in line with the camera in the real world. We use an area-based algorithm and will conclude a line is detected only when the bounding box is about 80% filled with black pixels. It turns out that it is very effective in determining a blackline on a white background later in the testing. After the

bounding box traversing is finished and a conclusion is made, it will write algo_done and algo_result to the slave registers to be processed by the processor.

```
assign algo_done = algo_en & (algo_count >= 8400); // 30*280
always@(posedge clk25)begin
	if( result == 0 ) begin
		if ( pixal_count >= 7500 && algo_en )
			result <= 1;
		else
			result <= 0;
	end
end
```

## 4.5 PmodDBH1

On the hardware side, we directly used the IP code that came with vivado IP library under vivado-library-master\ip\Pmods\PmodDHB1_v1_0.(Appendix C Github Link to Library Code) We include this IP to layout design and connect the physical dual bridge module to one of the Nexys4 pmod port.

On the software side, we started with the driver example code under vivado-library-master\ip\Pmods\PmodDHB1_v1_0\drivers\PmodDHB1_v1_0\src. Then we added our own function on top of it for driving the RC car. The details are provided in the basic driving part of section 4.7 Processor code.

## 4.6 PmodWIFI

On the hardware side, we directly used the IP code that came with vivado IP library under vivado-library-master\ip\Pmods\PmodWIFI_v1_0. We include this IP to layout design and connect the physical WiFi module to one of the Nexys4 pmod port.

On the software side, we started with the driver example code under vivado-library-master\ip\Pmods\PmodWIFI_v1_0\drivers\PmodWIFI_v1_0\examples\TCPEcho Server(Appendix C Github Link to Library Code). Then we modified it to communicate with a client on computer to get instructions on performing tasks getting line detection feedback, Maxsonar reading, and doing parallel parking. The details are provided in the TCP handler, part of section 4.7 Processor code.

## 4.7 Processor Code

### 4.7.1 Basic driving

The vivado-library-master\ip\Pmods\PmodDHB1_v1_0\drivers\PmodDHB1_v1_0\src(Appendix C Github Link to Library Code) driver code provides useful functions to control the H-bridge, including DHB1_setMotorSpeeds, DHB1_motorDisable,DHB1_motorEnable, DHB1_setDirs. Using these functions, we wrote our own functions in main.cc of drive_capture folder under auto_parking\auto_parking.sdk\drive_capture:

- drive function which enable the motor and then does usleep(100000) before disabling the motor. This moves the RC motor car forward for about 10mm. This drive function takes an input sensor_edges and drive the RC car a distance equal to 10mm x sensor_edges.
- drive_routine function which builds on drive function. This function also takes in two more inputs: dir1 and dir2. This routine call DHB1_setDirs with input direction and then call drive with input sensor_edges.
- Then the forward, backward, left, right function call drive_routing function with values for sensor edges, dir1 and dir2 to move the RC car in corresponding direction.

### 4.7.2 Line detection

On the software side, the line detection is the capture function in in main.cc of auto_parking\auto_parking.sdk\drive_capture. The function proceeds with the following slave register read/write sequences.

1. Write 1 and then 0 (button simulation) to slave_reg0 located at address myCam_ptr to start the hardware line detection algorithm.
   ```
   *(myCam_ptr) = 1;
   *(myCam_ptr) = 0;
   ```

2. Keep polling algo_done from the slave_reg2 located at *(myCam_ptr+2) until it is 1.
   ```
   while(*(myCam_ptr+2) == 0){
   }
   ```

3. Read the final line detection result from slave_reg3 located at *(myCam_ptr+3).
   ```
   int result = *(myCam_ptr+3);
   ```

4. Return the result.

## 4.7.3 Ultrasonic sensor feedback

vivado-library-master\ip\Pmods\PmodMAXSONAR_v1_0\drivers\PmodMAXSONAR_v1_0\src contains an API to query for the distance sensed by the ultrasonic sensors. To incorporate this function for our purpose: detect solid obstacles, we need to define the range at which an obstacle is considered "close". According to the specifications that can be found at https://store.digilentinc.com/pmodmaxsonar-maxbotix-ultrasonic-range-finder/, the minimum distance that the ultrasonic sensor can distinguish is 6 inches. I.e any obstacles 6 inches and within of the sensor will show a reading of 6 inches. Therefore, if the sensor reading is 6 inches, we will confirm that as obstacles detected while a reading of larger than 6 inches would be a not detected.

## 4.7.4 TCP handler

vivado-library-master\ip\Pmods\PmodWIFI_v1_0\drivers\PmodWIFI_v1_0\examples\TCPEchoServer provides example code to set up a TCP server using Pmod wifi. Building on this example, we added a TCP handler. Basically, the TCP server waits for input from a connected client and print the input value to SDK console. Depending on the input value, the TCP handler performs different mode of operation.

- For input value 0,1,16,17 , TCP handler calls forward, backward, left, right to test the motor control.
- For input value 2 and 3, TCP handler will call to test camera capture and sonar functions.
- For input value 34, TCP handler call to test parallel_park function.
- Importantly, for input value of 36, TCP handler will call line detection algorithm and parallel parking algorithm to park the car between two lines.
- For input value of 37, TCP handler will call sonar detection algorithm and parallel parking algorithm to park the car between two obstacles.

## 4.7.5 Auto-Parking Algorithm

The auto-parking algorithm involves the sonar function, the capture & line detection function, and the motor driving function. Depending on the scene setup, we have two separate auto-parking algorithms, one targeting parking between lines, the other targeting parking between objects. We will talk about each algorithm step by step in the following.

- Line algorithm
    1. Keep moving forward until the first line is detected.

```
int result = capture();
while(result != 1) {
      forward(1);
      result = capture();
      distance = sonar();
      usleep(100000);
```

```
}
```

2. Continue to move forward until the second line is detected. Keep a counter of the distance moved. Reset the counter if any obstacle is detected by the ultrasonic sensor during the process.

```
forward(2);
result = capture();
while(result != 1) {
      forward(1);
      Counter++;
      result = capture();
      distance = sonar();
      if (distance == 6) // minimum distance displayed
            counter = 0;
      usleep(100000);
}
```

3. Based on the counter value, determine if the spacing between two lines is sufficient to park the car or if any obstacle is present. If there is enough space, the car will proceed to perform a sequence of parallel parking into the spot autonomously; else it just stops and panics.

```
if (counter >= 20)
      parallel_park();
else
      xil_printf("Not enough space!\r\n");
```

- Object algorithm
    1. Keep moving forward until the car has moved out of the first object when ultrasonic sensor can no longer detect the object.

```
int distance = sonar();
while(distance < 8) {
      forward(1);
      distance = sonar();
      usleep(100000);
}
```

    2. Continue to move forward until the second object is detected by the ultrasonic sensor. Keep a counter of the distance moved.

```
forward(2);
while(distance >= 8) {
      forward(1);
      Counter++;
      distance = sonar();
      usleep(100000);
```

```
}
```

3. Based on the counter value, determine if the spacing between two objects is sufficient to park the car. If there is enough space, the car will proceed to perform a sequence of parallel parking into the spot autonomously; else it just stops and panics.

```
if (counter >= 20)
      parallel_park();
else
      xil_printf("Not enough space!\r\n");
```

# 5. Description of the Design Tree

https://github.com/JingJeremyXie/G2_AutoParallelParking
The github repository contains the following two directories: “doc” and “src”.
“Doc” directory contains the presentation slides and this report.
“Src” directory contains all the source code for the project, including our custom hardware IP and the final working Vivado project.
To compile the project, one must open the Vivado project file “src/auto_parking_final/auto_parking.xpr” in Vivado with the latest board files and Vivado IP library (available at the Xilinx website) added to the path, along with the custom IP in “src/ip_repo/myCam_1.0”.

Vivado Design Tree (with only major components listed)

- src/auto_parking_final/auto_parking.srcs/
    - sources_1/bd/design_1/
        - hdl/
            - design_1.v : block design verilog file
            - design_1_wrapper.v : top-level block design wrapper
        - ip/
            - design_1_axi_uartlite_0_0 : UART USB communication
            - design_1_blk_mem_gen_0_0 : BRAM generator for storing the picture
            - design_1_clk_wiz_0_1 : Clocking wizard
            - design_1_microblaze_0_0 : MicroBlaze processor
            - design_1_myCam_0_0 : custom camera+image processing module
            - design_1_PmodDHB1_0_1 : Pmod Dual H-bridge module
            - design_1_PmodWIFI_0_0 : Pmod WiFi module
            - design_1_PmodMAXSONAR_0_0 : Pmod Ultrasonic Sensor module
    - constrs_1/new/
        - cam_constr.xdc : constraint file for the custom camera IP
            - JA & JB Pmod ports
            - VGA outputs
            - Debug Switches
- src/ip_repo/myCam_1.0/
    - hdl/
        - myCam_v1_0.v : top-level wrapper file for custom camera IP
        - myCam_v1_0_S00_AXI.v : AXI lite slave IP implementation

    - src/
        - I2C_AV_Config.v
        - I2C_Controller.v
        - I2C_OV7670_RGB444_Config.v
        - ov7670_capture.v : ov7670 Pmod Camera image capture module
        - ov7670_top.v : custom camera top-level module
        - vga444.v : read image from frame buffer and process it to be displayed on VGA444, where line detection algorithm resides

SDK Design Tree (with only important files listed)

- src/auto_parking_final/auto_parking.sdk/
    - design_1_wrapper_hw_platform_1/
        - system.hdf
        - design_1_wrapper.bit
        - design_1_wrapper.mmi
    - drive_capture/
        - src/
            - main.cc : top-level main function of the software; send commands to Pmods
            - PmodDHB1.c : supporting functions for Pmod Dual H-bridge
            - PWM.c : pulse-width modulation for motor driving
            - PmodMAXSONAR.c : supporting functions for Pmod Sonar
    - drive_capture_bsp/
        - system.mss
        - microblaze_0/ : code library for running program on the MicroBlaze processor

Documentation

- doc/
    - ECE532_final_report.pdf
    - ECE532_presentation.pptx
    - Wiki page
        - Tricks and tips

# Appendix A Milestones From Proposal

Milestone #1:

| Team member | Responsibility |
|---|---|
| Cheng Lin He | Background Research<br>Explore car parking algorithm<br>Explore image analysis algorithm |
| Qiu Yuan Chen | |
| Jing Xie | |

Milestone #2:

| Team member | Responsibility |
|---|---|
| Cheng Lin He | Complete assembling the Lego Car<br>Research on Nexys 4 wireless communication<br>Research Memory configuration |
| Qiu Yuan Chen | Research Ultrasonic, Camera kit configuration |
| Jing Xie | Working on image processing algorithm |

Milestone #3:

| Team member | Responsibility |
|---|---|
| Cheng Lin He | Control the Lego car with Nexys board<br>Complete Memory Configuration<br>Complete the wireless communication |
| Qiu Yuan Chen | Get input from ultrasonic sensor<br>Work on auto-parking algorithm |
| Jing Xie | Work on image processing algorithm |

Milestone #4:

| Team member | Responsibility |
|---|---|
| Cheng Lin He | Get input from digital compass<br>Digital Compass Configuration<br>Work on auto-parking algorithm |
| Qiu Yuan Chen | Work on auto-parking algorithm<br>Get and analyze input from Camera<br>Camera configuration |
| Jing Xie | Integrate image processing algorithm with Camera input |

Milestone #5:

| Team member | Responsibility |
|---|---|
| Cheng Lin He | Integrate the whole project together<br>General Debug<br>Testing the auto-parking algorithm |
| Qiu Yuan Chen | |
| Jing Xie | |

Milestone #6:

| Team member | Responsibility |
|---|---|
| Cheng Lin He | Testing the project design<br>Modify the design if needed<br>General Debug |
| Qiu Yuan Chen | |
| Jing Xie | |

Milestone #7:

| Team member | Responsibility |
|---|---|
| Cheng Lin He | Demo and Design Fair |
| Qiu Yuan Chen | |
| Jing Xie | |

# Appendix B Weekly Accomplishment

Milestone #1:

| Team member | Achievements |
|---|---|
| Cheng Lin He | Background Research<br>Explore car parking algorithm<br>Explore image analysis algorithm |
| Qiu Yuan Chen | |
| Jing Xie | |

Milestone #2:

| Team member | Achievements |
|---|---|
| Cheng Lin He | Complete assembling the RC Car<br>Research on Nexys 4 wireless communication |
| Qiu Yuan Chen | Research Ultrasonic, Camera kit configuration |
| Jing Xie | Research image processing algorithm<br>Complete assembling the RC Car |

Milestone #3:

| Team member | Achievements |
|---|---|
| Cheng Lin He | Completed configuring Pmod Wifi.<br>Completed configuring H-bridge<br>Can drive the RC car motor with H-bridge and can control the speed and direction. |
| Qiu Yuan Chen | Get input from and configure ultrasonic sensor<br>Get input from and configure Camera |
| Jing Xie | Implemented the line detection algorithm in software (MATLAB)<br>Researched on H-bridge usage with motors and power source |

Milestone #4:

| Team member | Achievements |
|---|---|
| Cheng Lin He | Discussed possible ways of auto-parking algorithm<br>Solder motor and key component of the RC car<br>putting H-bridge, wifi pmod and RC car together |
| Qiu Yuan Chen | Wrote c code for Microblaze based on the optimal parallel parking technique in real life. |
| Jing Xie | Write a Verilog custom module to detect a line from the camera input (a picture taken and stored in memory) |

Milestone #5:

| Team member | Achievements |
|---|---|
| Cheng Lin He | Work on auto-parking Algorithm<br>Working on mid-term Demo |
| Qiu Yuan Chen | Work on auto-parking Algorithm<br>Working on mid-term Demo |
| Jing Xie | Integrate the whole project together |

Milestone #6:

| Team member | Achievements |
|---|---|
| Cheng Lin He | Developing the auto-parking algorithm<br>RC car can autonomously detect parking location between lines and perform a parking action |
| Qiu Yuan Chen | |
| Jing Xie | |

Milestone #7:

| Team member | Achievements |
| --- | --- |
| Cheng Lin He | Working on Demo and Design Fair |
| Qiu Yuan Chen | |
| Jing Xie | |

# Appendix C Github Link to Library Code

## Pmod DHB1:

https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodDHB1_v1_0/

## Pmod Wifi:

https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodWIFI_v1_0/

## Pmod MAXSONAR:

https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodMAXSONAR_v1_0/