Final Project: “ACDC”

Aria Lindberg

In association with: Will Gambero

Constructed: November-December 2024

Completion by: 12/12/24

Abstract

The following details the process, adjustments, and achievements made to Lafvin's 2-wheel drive car kits to optimize the vehicle. Nicknamed "ACDC" or "Automated Car using Device Control," ACDC is a modified version of the base model of the vehicle equipped with a Bluetooth communication chip and interactive headlights. This modified version allows a user to control the vehicle wirelessly using Bluetooth and an app GUI provided by RemoteXY, and as an extra addition also has semi-operational headlights. The result of implementing these functions pushes the hardware and software to some limitations, though the final product meets the specifications set before construction began.

Objective

The aim of the final project is to create a product using the tools and equipment discussed throughout the semester. This product should demonstrate that the lab participants' knowledge, skill, and creativity honed throughout the course has coalesced into a complex, thoughtful result. The base model of the vehicle includes two wheels and motors, a caster, two photodetectors, two line detectors, two IR sensors, an ultrasonic sensor with a servo, a battery pack, and an Arduino with an associated shield. In addition to these, a JDY-16 Bluetooth chip was switched for an HM-10 chip, which had better compatibility with RemoteXY and the Arduino, and several LEDs with a breadboard were added as mock headlights. The lab is comprised of several milestones for a minimum viable product: obtaining fundamental Bluetooth control and compatibility, implementing protection functions, and at least one other mode of functionality (like headlights). Anything obtained after these goals were considered part of the stretch goals. For more information about the project outline, consult the Project Proposal in the Appendix.

Procedure

Circuits

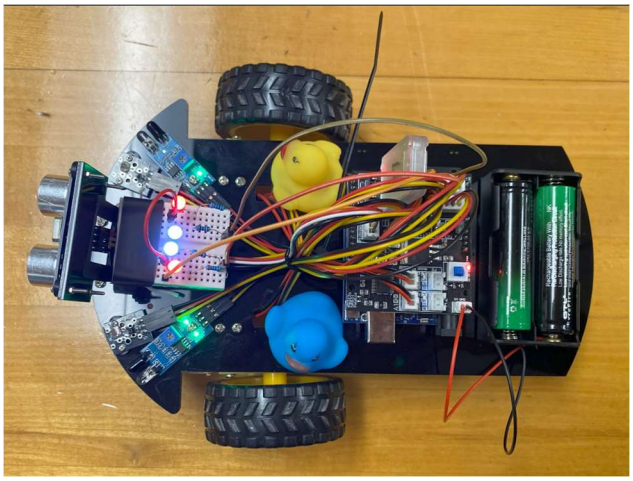


Figure 1: Top-View of Final ACDC

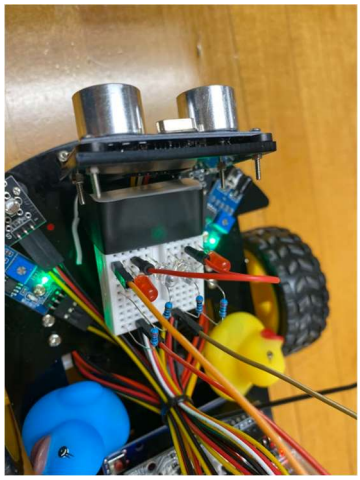


Figure 2: Isolated Headlights

Figures 1 and 2, above, display the finalized circuit, complete with the hardware connections. Not pictured is the caster and line detectors, which are underneath the vehicle.

One circuit is utilized for the vehicle's operation, provided primarily by Lafvin's kit and the central Arduino and shield. Figure 3 shows a complete schematic of the core vehicle aspects. However, after the addition of the headlights, the circuit expanded slightly to utilize empty pins for the headlight operation. This information is shown in Figure 4, which was made in LTspice. Figure 3 is courtesy of Lafvin, provided on their website with the vehicle.

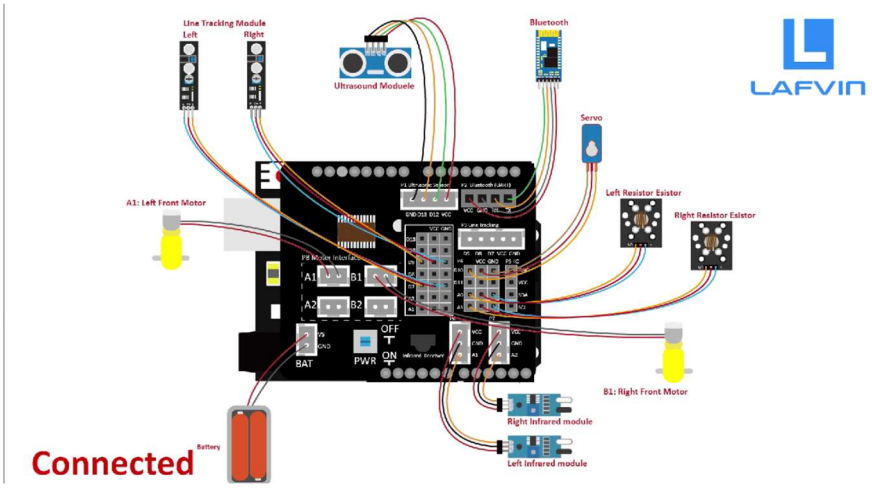


Figure 3: Core Circuit Schematic

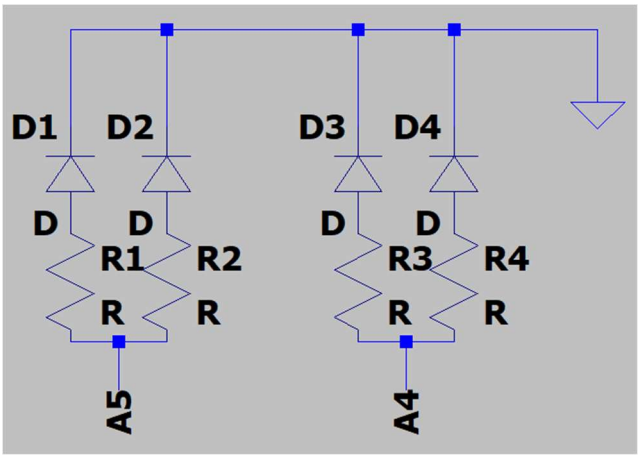


Figure 4: Headlight Circuit Schematic

In Figure 4, all resistors are 220 Ω, pin A5 connects to the righthand side headlights, and pin A4 connects to the lefthand side headlights. Diodes D_2 and D_3 are white LEDs, and diodes D_1 and D_4 are red headlights. If more pins were available on the shield or Arduino, these LEDs would have been isolated to operate more like a standard car's.

Results

The final product met the criteria for a minimum viable product as established in the project proposal. This included fundamental Bluetooth control, self-preservation functions, and at least one form of additional functionality which came in the form of the headlights. However, the stretch goal for a mobile GUI and Bluetooth control was also met, and gladly so as it brings new life into the project. For a detailed description of when and how the project progressed, consult the lab notebook featured in the Appendix.

The hardware list required for the final product included the fully assembled Lafvin car kit, with the exception of the line sensors, which were implemented but remained unused. It also required an additional breadboard with jumpers, four 220-Ω resistors, and two red and two white LEDs for the headlights. The JDY-16 Bluetooth chip designed for the car kit was omitted and replaced with an HM-10 chip, as the HM-10 is compatible with RemoteXY for implementation of the GUI software elements. Since RemoteXY is an application, a cellular device with the capacity to download applications is also required, although both Aria and Will were able to operate the ACDC with their phones. To note, Will uses an Android, and Aria uses an iPhone, both of which were compatible with RemoteXY. A screenshot of the GUI is shown below in Figure 5.

Figure 5: RemoteXY GUI Displayed in App

The GUI features a speed slider to adjust the speed of the ACDC, an emergency stop button which stops the car when pressed and keeps it stopped when held, and a joystick to direct the ACDC in a given direction.

The function of the car is isolated to one main code file, though to explain and test different components, aspects of the ACDC's function were split into subsections prior to committing them to the main file. These subsections, and the main code file, can be accessed by the Github link provided in the Appendix. The functions of different parts of the code are generally included in the comments of the programming files, unless otherwise specified. The main file is "BluetoothPlus_MobileDeviceCarContol," and the others are included as commits prior to adding them to the main file.

The first section of the code consists of definitions, includes, initializing variables, and setting pin values. First, the #defines and structure provided by RemoteXY are included, which are the bridge for communication between the Arduino and Bluetooth to the phone application inputs. In this section are the preambles to the maps performed later on, but these predetermined values are set by RemoteXY for specific ranges. These include the functions of the joystick, speed slider, and emergency stop, which are commented with their ranges.

The second section of the code is initializing and setting hardware connections in order. This includes: head and trail lights, wheel control, ultrasonic and servo, photoresistors, line trackers, and IR sensors. Following these are constant and tracking variables which are used in the logic, and each of these variables are explained within the comments of the code.

The third section of the code are the built-in functions, established in the void setup. This includes setting a number of pin modes using pinMode(), attaching the servo, beginning the Serial monitor, and initializing the RemoteXY library functions from the first section of code. The majority of the pinouts are either outputs or pullup inputs, as described in the void setup.

The fourth, and arguably most important section of the code is contained in the void loop. In the void loop, looping constantly, are checks for other modules which are later established. This includes a RemoteXY handler, direction mapping from the joystick to the wheel motors, a hazard check for potential front hazards, a check for ambient light levels, and a check for the status of the headlights. What follows in the void loop is an on-off sequence for the flashing headlights composed of if-else statements. The "Flashing_Threshold" set at 2000 is set at 2000 due to complications posed by including a delay. By using if-else statements, a loop is created which continues until the threshold is met, thereby creating a manual loop to keep the duration that the lights are on and off at a visually-good rate. Other notes on the behavior of these if-else statements are included as comments in the code. The last portion of the void loop contains a very important aspect of the ACDC's safety features: the emergency stop button. The emergency stop button needs to be accessible, per the name, and therefore must be accounted for at all times. It is implemented simply; with a single if and a single else statement. If the button is pressed and held down, the car remains stopped and the headlights are all turned on, as if the breaks were hit. All lights are turned off in the else statement, unless the ambient light has detected darkness in which case they remain on.

All of the following code sections are customized functions each designed to perform a specific task which does not need to be isolated in a void setup or void loop. However, many of these functions are called within the void loop.

The fifth section of code is the DirectionConverter function which utilizes other customized motion functions (elaborated on later) to toggle movement of the ACDC. If the emergency stop button is not pressed, then the program will wait for input of the speed slider and joystick to reach certain thresholds (set through trial and error performance testing) before moving. The series of if-else statements allows full range of motion of the ACDC so long as the requirements of said statements are met. These requirements are described more in depth as comments within the code.

The sixth section of code is the AmbientLightCheck function which tests for ambient light levels using the photoresistors. The readings of the photoresistors are compared to a "Nighttime_Threshold" which determines if it is dark enough to warrant the headlights being on. Nighttime_Threshold was determined via trial and error performance testing. If the threshold is met, the headlights are turned on. If not, they remain off.

The seventh section of code is the BlinkerLightsCheck function which tests for and implements a turn signal light mode. When the joystick value for the x-axis direction reaches thresholds specified by "Blink_Threshold," the lights for the corresponding direction will turn on. If the x-axis value is too small, then the lights are left off.

The eighth section of code is the FrontHazardCheck function which tests for front-facing hazards and implements an auto-stop feature. If either IR sensor sees an obstacle around 4 inches

in front of the ACDC, it will automatically stop the car and turn on the headlights. The distance is set manually by adjusting the screws on the potentiometers of the IR sensor chips.

The ninth, tenth, and eleventh sections of code all correspond to wheel and direction functions. The MoveForwards and MoveBackwards functions take two inputs, TurnRatio and Speed, which are taken as inputs from the RemoteXY GUI. These values are then mapped and included in the analogWrite to the wheel motors, adjusting the speed and direction. The Stop function simply stops the ACDC by analogWriting values of 0 to the wheel motors.

The collection of these functions and sections provides a full range of motion annotated with headlight commands, as well as safety features to maintain the quality of the ACDC.

Conclusion

The final project, ACDC, meets its criteria specified by the project proposal and then some, operating well with the Bluetooth GUI, maintaining the safety features, and using headlights as additional navigation assist. The end product is satisfactory for its intended purpose— a device designed for fun, and a great testament to the challenges posed by implementing so many features. In future iterations, it may be wiser to make a custom PCB for further development of accessories and stretch goals.

Appendix

Additional Figures, Programming, & Lab Notebook:

All programming can be found in the Github with the following link:

https://github.com/HiWillCat/BluetoothPlus-Car-CMPE3815/tree/main

Project Proposal:

Final Project Proposal: Bluetooth+ Car Control

Lab Group: Aria Lindberg & Will Gambero
University of Vermont, Department of Electrical Engineering
CMPE 3815: Microcontroller Systems

Concept:

For the final course project, we will construct a Bluetooth- controlled car. The bare minimum is to send letter commands to control standard movements–these include turning left/right, going forward/backward, and stopping. We want to next upgrade this to a mobile GUI to control the car similar to a remote control with buttons; we will also look to introduce extra functionality to sensors such as daylight-sensing headlights.

Objective:
Construct a program to control a miniature car using Bluetooth communication to perform fundamental movements via single letter inputs as commands; this is planned to be stretched to mobile UI to replace text commands with interactive buttons. This will be easily verified with trials using a mobile device to send commands and later through GUI button pushes.

A. Hardware List
The required hardware components are:
- Arduino Uno
- Constructed car (car kit)
  - Servo, ultrasonic sensor, DC motors, line trackers, IR sensors, photoresistors, plastic parts, batteries, etc.
- Bluetooth hardware
  - Receiver, mobile device
- Extra breadboard
- Wires/cable connectors
- Resistors
- Assorted LED lights
- Screwdrivers, spare screws

B. Milestones
The series of tasks to say the final project was a success will be split into two important categories:

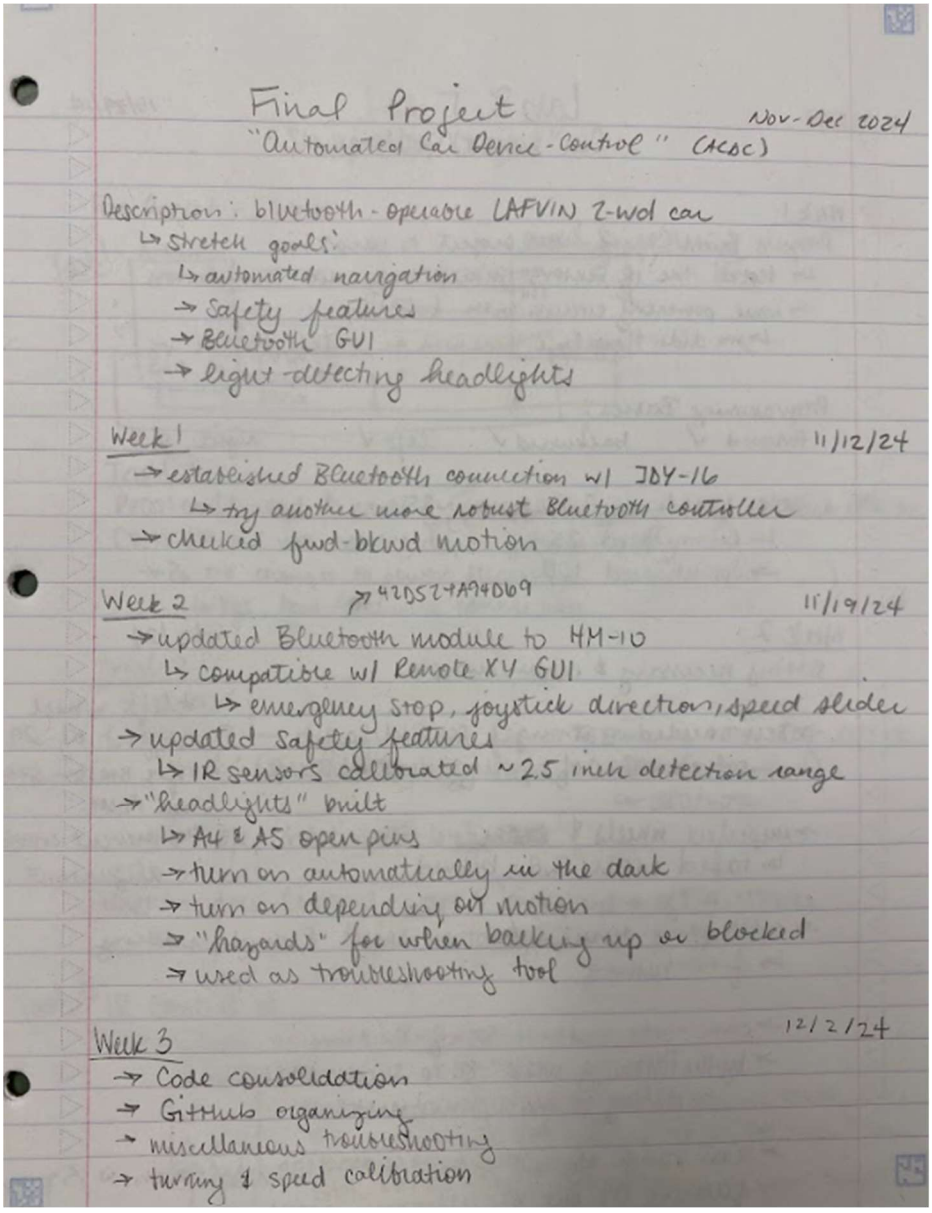
MVP (minimum viable project) – the minimum requirements to deem the project successful:
- Fundamental Bluetooth control
  - Send, receive, and interpret single letters as movement commands.
  - Implement Bluetooth receiver into car kit shield
- Protection functions
  - Add internal functions to minimize the chance of the car hitting walls or edges.
    - Utilize line trackers, ultrasonic, or IR sensors
- Extra car functionality
  - Implement addition functionality to car; some ideas include:
    - Night-time LEDs
    - Horn with speaker/buzzer
    - Reversing lights

Stretch Goals – tasks to hopefully complete as extra once MVP is successful:
- Mobile GUI + Bluetooth control
  - Create mobile GUI to replace letter commands with interactive buttons
    - Similar to standard RC car remote controllers
  - Research into websites that offer GUI creation for Bluetooth
- Toggleable "auto-pilot" mode
  - Add mode to have car auto-move forward & navigate obstacles
    - Toggle/untoggle via extra GUI button

Questions:
- How much data can be sent through Bluetooth?
- What software can allow for custom Bluetooth GUI? How does Bluetooth GUI work?
- How/what libraries will be required to interpret Bluetooth UI commands?

Figure 6: Project Proposal

Lab Notebook:



Final Project — Nov-Dec 2024

"Automated Car Device-Control" (ACDC)

Description: bluetooth-operable LAFVIN 2-wd car
- ↳ Stretch goals:
  - ↳ automated navigation
  - → Safety features
  - → Bluetooth GUI
  - → light-detecting headlights

Week 1 — 11/12/24
- → established Bluetooth connection w/ JDY-16
  - ↳ try another more robust Bluetooth controller
- → checked fwd-bkwd motion

Week 2 — 11/19/24 — 42052+A94D69
- → updated Bluetooth module to HM-10
  - ↳ compatible w/ Remote XY GUI
    - ↳ emergency stop, joystick direction, speed slider
- → updated safety features
  - ↳ IR sensors calibrated ~2.5 inch detection range
- → "headlights" built
  - ↳ A4 & A5 open pins
  - → turn on automatically in the dark
  - → turn on depending on motion
  - → "hazards" for when backing up or blocked
  - → used as troubleshooting tool

Week 3 — 12/2/24
- → Code consolidation
- → Github organizing
- → miscellaneous troubleshooting
- → turning & speed calibration

Figure 7: Lab Notebook (Aria)