Final Deliverables

Summary of Lessons Learned

- The Kanban served a good purpose for initially segmenting what different possible objectives would be relevant in the completion of this project.
- Using a GitHub repository to host our project files enabled a centralized environment that made it easy to share and update our workflow.
- There are many external factors that can affect the development and testing of a project, so adapting quickly to these allows us to make efficient use of our time. (Car mounting points breaking, servo connection breaking, car uncharged, camera driver issues aboard Debian, hardware failure and threshold).
- In-person collaboration showed the most efficient usages of our time, as we were able to quickly bounce back feedback and help each other with issues we struggled with.
- Using Discord, which most of us already use regularly, allowed us to maintain contact with quick response time and high availability. It also permits us to quickly plan meetings and contact each other with questions that have ample response time.
- We had to be adaptable to learning new things within our code and our project as well. Learning the correct values and continually manually testing our code, either through hardware + code or solely code input itself let us learn a lot regarding debugging. It allowed for a linear and methodological way to test our software and the way it interacted with the hardware.

Planned v. Actual Accomplishments

Planned:

- Rough idea of our original plans involves a local hosted site to house a live feed recorded by our Beaglebone. This site would also contain indicators that show input from our controller, represented visually.
- We planned to use the controller connected to our local client (laptop), where we'd somehow send the input to the site, as well as to the motors that controlled the car.
- Controller inputs were to be mapped to left and right triggers, with acceleration and deceleration mapped to them respectively. Steering would be mapped to the left joystick. (Mimics the layout of vehicle controls in video games)

Actual:

- Controller to laptop to BeagleBone connection correctly initiated
- Values from controller mapped to responsive local GUI application built in PySimpleGui
- Controller input translated to our Beaglebone in order to operate motors
- Motors are correctly armed and translate the values correctly respective to given boundaries of servo and controller.
- Webcam capture is able to be captured live from the BeagleBone and viewed from the local client.
- Controller inputs are correctly mapped, with inclusion for increasing the power threshold to our motors.

Our final product is in line with our development plans, with a few tiny changes here and there that did not deviate too much from the deliverable. Brainstorming everything we'd need ahead of time allowed us to account for most of the variables that we'd have to deal with.

As-Built Current Design Implementation

- Local client (Laptop) runs controller and UDP connection python files.
- Controller sends values through UDP socket to server hosted aboard BeagleBone
- Values are translated into the two servo motors to arm and operate the steering and powertrain servo motors.
- BeagleBone hosted webpage is started for webcam, where it's accessed and viewed on our local client.
- Provided Race management code sends a signal in order to enable Servos and start a camera connection to RM's RTSP server.

Description and Inclusion of Code with Instructions to Install/Run

- All files are zipped in racer1_002.zip
- Source code is split into two separate folders: clientSide and serverSide.
- clientSide folder requires the following libraries installed on the laptop side: socket, json, pygame, time, PySimpleGUI, socketio.
- serverSide folder requires the following libraries installed on the beaglebone black side: opencv, pyshine, adafruit_BBIO, socket, json
- To run everything, do the following:
    - Connect your PS4 controller to your laptop (either wired or bluetooth)
    - Have your microSD card inserted (with serverSide files) to beaglebone.
    - Plug in the beaglebone to a battery source
    - Plug in the USB hub into the beaglebone with wireless adapter and webcam attachments.
    - Plug appropriate pins from the RC car into the beaglebone.
    - Ensure the beaglebone is connected to the wireless modem via:
        - Connecting to the modem on your laptop
        - Using connmanctl to connect to the modem on the beaglebone
        - Finding the wireless IP address for the beaglebone using ifconfig
        - Opening the beaglebone's IDE using the wireless IP address in a separate tab.
    - Turn on the motor on the RC car. You will get two beeps.
    - Modify the server_UDP.py code to use the same wireless IP address gained using ifconfig. Run the server_UDP.py file.
    - Run the camera.py file to get a http link to start up the camera on a separate tab.
    - On the laptop side, modify the client_UDP.py file to have SERVER be the wireless IP address of the beaglebone.
    - Run controller.py and wait for race management to accept the race car
    - Press "X" to start up the motor and steering. This will give you the third beep on the ESC.

    - Afterwards, everything should be connected and you will be able to control the steering of the car with the left stick, and the acceleration and deceleration with left and right trigger, respectively, as well as a forward and reverse boost with R1 and L1.
    - You can exit the program by pressing “Options,” or pressing “Triangle” to close the controller.py file but leave the server up and running.

Full Testing Reports - Metrics/Methodology/Code Coverage/Scope/Open+Closed System Defects

- Our controller was tested via interactive/functional testing, where we initially would all collectively take turns as a group pressing buttons on the controller to observe the expected behavior of the buttons translating to their correct values on the computer.
- Additionally, when we implemented the servos functionality by using the left analog stick, we would take turns moving around the stick while one person would carefully inspect the generated servos angle on the computer to ensure it was in range.
- When we implemented the R2/L2 functionality, we would take turns pressing R2/L2 at different intensities while one person would carefully inspect the generated accel/decel value that was generated by pressing the button at any given intensity.
- When we were finished implementing the throttle mechanic of our racecar by using R2 and L2 of the PS4 controller, we extensively tested the performance of our vehicle with different translated ranges of throttle from the starting values of: [-1, 1]. This was done by simply holding and alternating R2 and L2 to ensure that the middle value of our range would stop any acceleration, while the higher and lower ends of our range would propel the car backwards and forward respectively.
- Regarding the servos angle values, we also tested these with the actual racecar for any sensitivity issues regarding the left analog stick movements that may have been needed to fix, such as the range of servos angles that would be appropriate for steering/handling.
- Since our controller functionality was already tested extensively in real-time, we really only needed to write unit tests for smaller functions that we used in our controller.py source file when actually using the controller.
- We wrote unit tests for getStickAngleAlt() to ensure that each of the bounds and 0 would return the correct stick angle for their respective x-axis parameter input value.
- We also wrote unit tests for update_last_value() to ensure that the logic works properly to send the correct value to the beaglebone based on if the last value is over the threshold or not. This unit test uses some sample cases and asserts the expected output. These sample cases are some that may be similar to actual values that may be passed in to this function during execution.
- Our code coverage for these unit tests reaches every branch and statement at least once, where one case in update_last_value() even tests an edge case that is true.
- Overall, we have tested all the functionality in controller.py by functional, unit, and interaction testing while working on this racecar project.