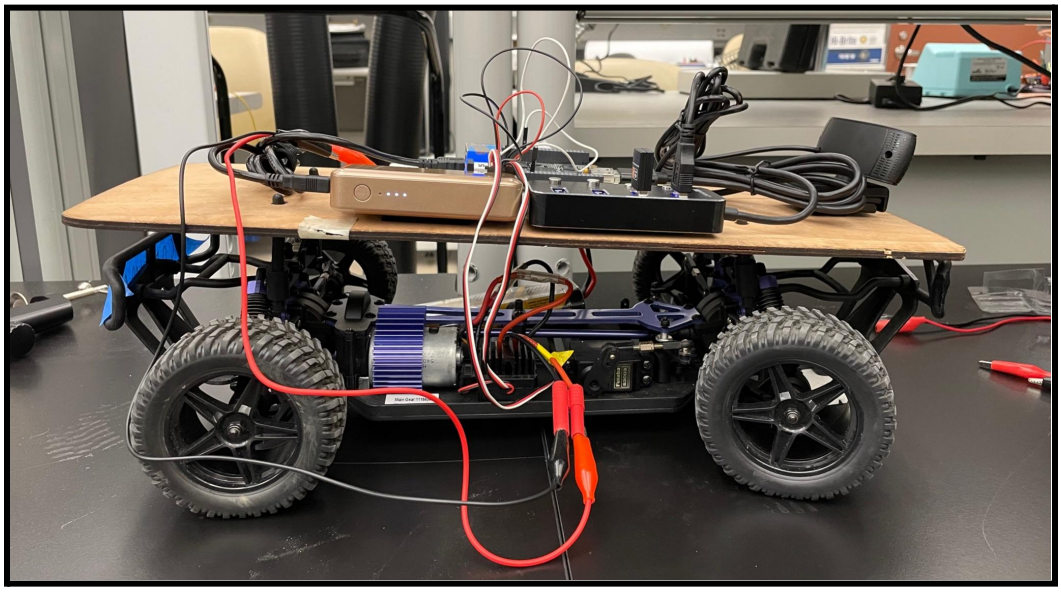
ELEC 424 - Final Project

200 points

“Ask any racer. Any real racer. It don't matter if you win by an inch or a mile. Winning's winning.” - Dominic Toretto



Overview

Autonomous vehicles are all the rage. Tesla, Rivian, Waymo, …. you name it. It’s an exciting time, and we don’t want to miss out on the action. What could be cooler than building your own autonomous vehicle?

For the final project of 424 you will work on a team to program the brains of an autonomous RC/toy car. If it’s autonomous, then of course RC (remote control) loses its meaning, but if we say toy car then it might be ambiguous. So, we’ll say RC car often just because it implies a small toy car with electronics.

To make this happen, you can use the code from last year as a starting point: link [You can literally start from their scripts, just be sure to cite your sources; I had the most luck getting this one to run after removal of code that doesn’t matter for us this year, however I didn’t personally try the performance of it]. Those scripts take an existing Python script meant for a Raspberry Pi attached to an RC car and modify it to work with the Beaglebone Black (BBB) attached to the RC cars in Ryon. Your job is to make this system work on a BeagleBone AI-64 (BBAI64) along with using speed encoding. You will not only enable the car to perform lane keeping, but also to stop at a red stop box on the ground and (if you are a graduate team) to recognize a stop

sign using machine learning and stop. Grading will be based primarily on the performance of your car on the track laid out in Ryon B12.

Important note - P9 of the BBAI64 is different than that of the BBB! It has 4 extra pins at the top that have “E” labels. Then, the normal indexing of P9_01, P9_02, etc., occurs. Take this into account! P8 is the same between the BBAI64 and BBB. I made an embarrassing forum post about this. The Internet wins again.

As my life coach and personal mentor Bane said, “Let the games begin”.

Rubric

1. **(120 points) In person demonstration of functionality to instructor or TAs. We’ll coordinate a time with you if you cannot make it to our office hours. A video of your car performing this demonstration without the instructor/TAs present can earn up to 80 points, scaling the requirements from 120 points to 80 points.**
    a. **(40 points) [30 points if graduate team] Speed encoding**
        i. (20/15 pts) System uses speed encoder to maintain a somewhat consistent speed
        ii. (20/15 pts) System implements driver to precisely measure timings between optical encodings and these timings are used to adjust the pulse width modulation (PWM)
    b. **(40 points [30 points if graduate team]) Lane keeping**
        i. (25/20 pts) Car is able to follow the center of the lane for half of the track
        ii. (25/20 pts) Car is able to follow the center of the lane for all of the track

    Note: I define **following** as any edge of the car not moving more than 8 inches away from the lane marker closest to the car. Ask me if you are encountering problems with staying this close to the lanes.
    c. **(40 points [30 points if graduate team]) Red stop box on ground**
        i. (30/20 points) Car stops at first stop box, then resumes lane keeping
        ii. (30/20 points) Car stops permanently at second/final stop box
    d. **[30 points if graduate team] (not required for other teams)**
        i. (20 points) Car stops at stop sign
        ii. (20 points) Used machine learning to perform stop sign detection
2. **(40 points) Hackster page on car development, performance, and PID tuning - include the following elements in a project submitted to this community.**
    a. (2 points) Name
        i. Put the team name here that you created
    b. (2 points) Cover image
        i. Exercise your creativity on your choice of an image

c. (2 points) Elevator pitch
    i. 1 sentence on what your system does
d. (2 points) Components and apps
    i. Include 4 components (webcam, BBAI64, USB WiFi adapter, portable charger) and 1 app (openCV)
e. (2 points) Teammates - please have team members' Hackster accounts linked to the project so that they display on the page
f. (26 points) Story
    i. (2 points) 1-3 sentences introducing the project and what the car does. You must cite that this work draws from the following Instructable:
        - User raja_961, "Autonomous Lane-Keeping Car Using Raspberry Pi and OpenCV". Instructables. URL: https://www.instructables.com/Autonomous-Lane-Keeping-Car-Using-Raspberry-Pi-and/
        - Also be sure to cite any existing 424 projects on that site that you used
    ii. (4 points) 3-5 sentences about how you determined the resolution, proportional gain, derivative gain, and (optionally) integral gain to use for the car.
    iii. (4 points) 3-5 sentences about how you handled the stop box and (if graduate team) an additional 3-5 sentences about how you handled the stop sign.
    iv. (4 points) On one plot show error, steering PWM, and speed PWM versus frame number for an entire run of the track
    v. (4 points) On one plot show error, proportional response, derivative response, and (optionally) integral response versus frame number for the same entire run of the track as used for the previous bullet point.
        - The proportional response is the proportional gain times the error, the derivative response is the derivative gain times the derivative of the error, and (optionally) the integral response is the integral gain times the integral of the error
    vi. (4 points) Picture of one of the vehicles with your hardware on it
    vii. (4 points) Video of a vehicle completing the course with your hardware and software running on the vehicle
g. (4 points) Code
    i. Please upload any .py files and/or other code used for the project
        - You must include your modified device tree source file (the PWM file) and driver source file
    ii. You must cite the Instructable that the main .py file comes from at the top the file:

- User raja_961, “Autonomous Lane-Keeping Car Using Raspberry Pi and OpenCV”. Instructables. URL: https://www.instructables.com/Autonomous-Lane-Keeping-Car-Using-Raspberry-Pi-and/
- Also be sure to cite any other existing 424 projects on Hackster that you used

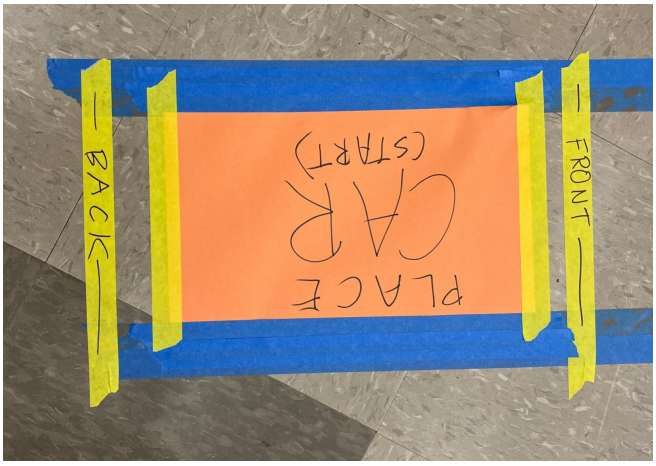
3. **(20 points) Evaluation of teammates’ effort**
    a. (5 points) Fill out a form estimating teammates’ effort
    b. (15 points) The average of teammates’ estimates of your effort

    Note: The evaluation will be about whether or not everyone did at least their fair effort.
4. **(20 points) Submission of relevant commented source code files to Canvas**
    a. Code attempts to achieve objectives/requirements stated in instructions
    b. Code reasonably includes comments
    c. The following file(s) must be submitted in their source form (e.g., .py, .c, .dts) - not a PDF
        i. Any .py file(s) and/or other code used for the project
        ii. You must include your modified device tree source file (the PWM file) and driver source file
        iii. We should be able to rerun your python code, don’t just give us a file that has functions without the script that calls those functions

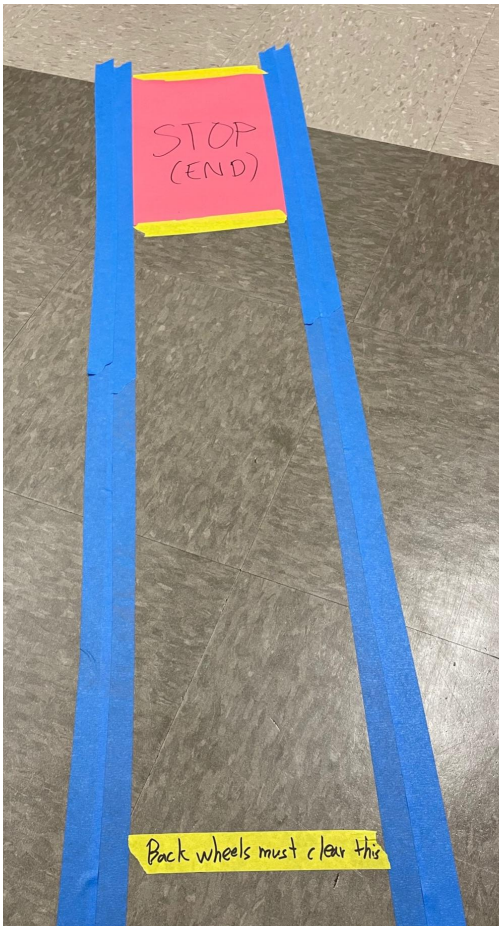
Guidelines

- **General**
    - Back up your code frequently! Your BeagleBone AI-64 could get fried. If you break it, you do not get another one (we don’t have anymore). You will have to make the project work on a BeagleBone AI, which I have some extras of.
- **Track**
    - The following is a picture of the track. The start is on the left, and the end is on the right. Your car must successfully navigate the track, stopping at the stop boxes (red paper on the ground) and (if graduate team) addressing a stop sign using machine learning. The car should completely stop at the second/final stop sign.

- The following picture shows the start where the car must be placed



- The following picture shows the end where the car must stop - specifically, the car must stop after the yellow tape saying “Back wheels must clear this”



- Note the “stop box” - it’s a red piece of paper on the ground :)
- Similarly, there is a red stop box in the middle of the track as well

- The stop sign is printed on a piece of paper and can be mounted to a box to be at the height and distance from the track you want it to be. Put it anywhere

around the yellow tape on the ground that says TRAFFIC LIGHT (last year we used a “traffic light”)
    - The rest of these guidelines detail requirements and advice around lane keeping, the stop boxes, and the stop sign
- **Connect to Rice Visitor**
    - Log in to the BBAI64
    - Run the command:
        - `sudo nmcli connection add ifname wlan0 type wifi ssid “Rice Visitor”`
    - Then do: `sudo reboot now`
    - Your BBAI64 should automatically connect to Rice Visitor from now on
- **Code to start with**
    - Start with one of the 424 project files here and/or the lovely Python file provided on Instructables (Autodesk, Inc.) here by user raja_961
- **Pulse-width modulation (PWM)**
    - The existing Python files use either an Adafruit PWM BBB library or a Raspberry Pi PWM library
        - These libraries don’t work with the BBAI64 - aren’t you feeling lucky? ;)
    - For the BBAI64 (based on the instructions here from user RobertCNelson on beagleboard.org), do the following commands to get PWM available:
        - cd /opt/source/dtb-5.10-ti-arm64/
        - git pull
        - make
            - [NOTE: You can ignore the warnings]
        - sudo make install
    - Then you need to do “sudo nano /boot/firmware/extlinux/extlinux.conf” and add the text “fdtoverlays /overlays/BONE-PWM0.dtbo /overlays/BONE-PWM1.dtbo /overlays/BONE-PWM2.dtbo” [which is 1 line] as a new line after the line “fdtdir /” but before the line “initrd /initrd.img”
    - Finally, reboot using:
        - sudo reboot
    - Following RobertCNelson’s instructions: Once your BBAI64 has rebooted and you are logged in again:
        - Verify that the PWM overlays loaded by running: sudo beagle-version | grep UBOOT
        - The output should look something like (bolded part important):
            - UBOOT: Booted Device-Tree:[k3-j721e-beagleboneai64.dts]
            - **UBOOT: Loaded Overlay:[BONE-PWM0.kernel]**
            - **UBOOT: Loaded Overlay:[BONE-PWM1.kernel]**
            - **UBOOT: Loaded Overlay:[BONE-PWM2.kernel]**
        - This means PWM is accessible, now we want to use it

- Here is how you should initially set up PWM using Python (you must run these lines anytime you reboot):

```
# P9_14 - Speed/ESC
with open('/dev/bone/pwm/1/a/period', 'w') as filetowrite:
    filetowrite.write('20000000')
with open('/dev/bone/pwm/1/a/duty_cycle', 'w') as filetowrite:
    filetowrite.write('1550000')
with open('/dev/bone/pwm/1/a/enable', 'w') as filetowrite:
    filetowrite.write('1')

# P9_16 - Steering
with open('/dev/bone/pwm/1/b/period', 'w') as filetowrite:
    filetowrite.write('20000000')
with open('/dev/bone/pwm/1/b/duty_cycle', 'w') as filetowrite:
    filetowrite.write('1500000')
with open('/dev/bone/pwm/1/b/enable', 'w') as filetowrite:
    filetowrite.write('1')
```

  - Notice, we are accessing PWM through the /dev interface
  - The first three lines set up pwm1a with a period of 20 ms (50 Hz), a duty cycle of 1.55 ms (roughly 7.75%), and enable the PWM
  - The second group of three lines set up pwm1b with a period of 20 ms (50 Hz), a duty cycle of 1.5 ms (7.5%), and enable the PWM
  - Strangely, you must enable both a and b PWMs before either will work
  - I recommend putting these lines in a dedicated python file and running it as sudo (e.g., sudo python3 init_pwm.py)
  - With this setup, you can now modify duty_cycle (as done above) anytime in your code
- Both the electronic speed controller (ESC) for controlling the speed of the car and the servo motor for steering the car do nothing (0 speed, straight) at a duty cycle of 7.5% (although I've been finding it can be higher for some cars for the ESC, hence why I used 7.75% earlier) for a PWM frequency of 50 Hz
- You will need to make sure to initialize the ESC anytime you power it on
  - Have a BBB PWM pin connected to the the ESC with the pin outputting a 7.5% (or 7.75%) duty cycle PWM waveform at 50 Hz
  - Then connect the ESC to the battery
  - You should hear two beeps with a few seconds in between them
  - The second beep means that the motor is ready to go
    - The ESC uses the duty cycle you give it during this period as the baseline (i.e., no movement at this duty cycle)
  - Increasing the duty cycle will move the car forward - please make sure to not go above 8% duty cycle for the ESC - the car will really take off above that and likely break something when it hits a wall
- Initialization is not needed for the steering servo

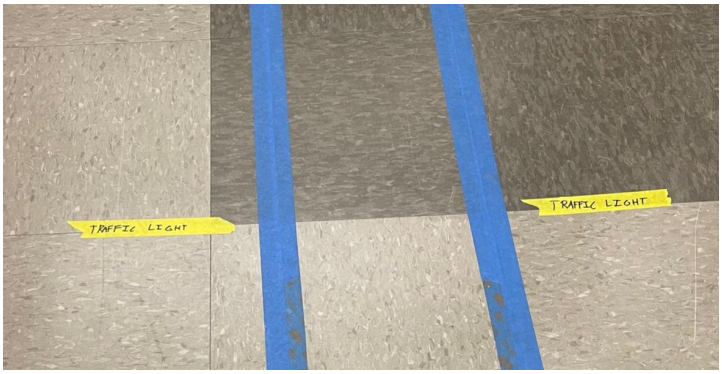
        - 7.5% duty cycle at 50 Hz will cause the servo to be straight
        - 6% will turn one way almost fully
        - 9% will turn the other way almost fully
    - Note that the Instructables code just turns the car left or right in a binary fashion
        - This will likely not be good enough for the course
        - I got things working by keeping the speed fixed low and using PD (proportional derivative control) with the steering
        - **You should only run the core loop of the code (the loop that performs lane detection and PWM) for a limited number of times - otherwise the speed motor will keep on going even when you ctl+c the running Python code! This means you also need to add a line after that loop that sets the motor duty cycle back to 7.5% so the ESC stops moving the motors.**
        - **As a backup, you can (1) have a Python function that sets PWM outputs back to a duty cycle of 7.5% and (2) disconnect the 7.2V battery for the car when needed. I've had to pick up the car sometimes and do one of these two since it was still running. You could also turn the switch off the ESC.**
- **Lane keeping**
    - You will generally interact with the car by wirelessly ssh'ing into the BBB attached to it. By doing so, you can wirelessly tell the car to start its Python lane keeping/self driving script. If you ssh in with X as I mentioned in class, you will also get a computer vision stream to your laptop. I recommend commenting out all of the image showing functions in the script except for the one that shows the lane detection and intended path.
    - You will need to resize the image in order for processing to occur at any reasonable rate
    - This is one of your parameters - it will have to be fairly low, even 720p is too high
        - You can get away with a pretty low resolution here, try out different stuff
        - You can set the webcam to run at a lower resolution using opencv, the following command shows you what resolutions your webcam supports:
            - v4l2-ctl -d /dev/video2 --list-formats-ext
        - Then you can use opencv to resize it even smaller
- **PID/PD controller for steering**
        - The current code implements a PD controller for steering
        - You may need to change the proportional and derivative gains
        - You have to think about how the error gets mapped by the PD controller to action (steering)
        - Once aspect of this is the range of values that will be output by the PD controller

        - The PD controller should really only cause the steering PWM to go between 6% and 9% (it would produce a PWM of 7.5% for zero error)
        - This is why I put a plotting requirement in the rubric for the report - it is helpful to plot out different aspects of the error/PD/steering and see how they are interacting
        - You can start out by setting P gain to some low value and D gain to zero
            - As you increase the P gain, the car should be more responsive to error, but eventually it will oscillate around the desired trajectory
            - Increasing the D gain allows you to reduce this oscillation
        - By plotting and testing, you should be able to find values that get you there
        - I highly encourage you to carefully observe the car's behavior and sometimes plot things to figure out how to work with P and D, rather than just guessing values (trust me, I've done the latter a lot…)
- **Encoder-based controller for speed**
    - The optical encoder will output high or low depending on whether or not light can shine through the wheel
        - Note - the encoder has 3 pins:
            - 5V - connect this to 5V of your BBAI64
            - GND - connect this to ground
            - OUT - connect this to pin P8_03 of your BBAI64
    - You can modify the project 3 driver to count the time between optical encoder activations, since the optical encoder will act like a button
    - This requires modifying the device tree of the BBAI64 to set up a pin (let's say P8_03) as the input reading the optical encoder
        - You want to add this to the end of the /opt/source/dtb-5.10-ti-arm64/src/arm64/overlays/BONE-PWM0.dts file:

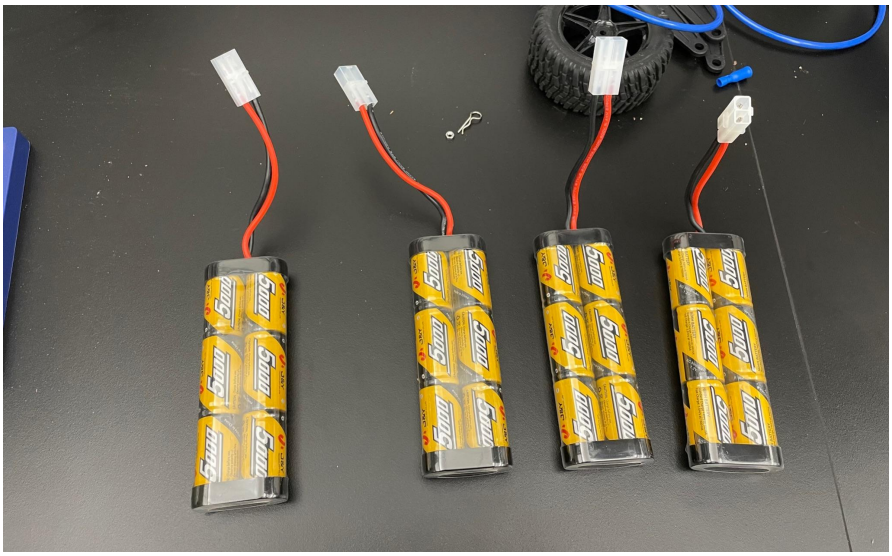
```
&{/} {
	gpio-leds {
		compatible = "hello";
		userbutton-gpios = <gpio_P8_03 GPIO_ACTIVE_HIGH>;
	};
};
```

        - Then you will follow a similar process to the PWM enabling:
            - cd /opt/source/dtb-5.10-ti-arm64/
            - make
            - sudo make install
            - sudo reboot
    - Then your driver should be able to recognize changes in the optical encoding as button presses, and you can add extra code to your driver to calculate the timings

        - Reject timings below 1ms, those are issues in the encoder like switch debouncing
        - I added capacitors next to the supplies that you will collect; You should grab 1 capacitor to put between ground and the OUT output of the encoder, since this will help “debounce”
        - I also changed the gpiod_set_debounce second input argument to 1000000 for time in microseconds. Seems wrong, but works well enough. Not sure why!
    - Somewhere in your main python script that is handling the system, you will want to periodically find out the rate of turning that your button/encoder driving is recording
        - You could get this value by adding some of the file operation support in your driver that we did previously in an exercise, then you would be able to open and read a “file” in python that reports the rate of turning, and take PWM action based on that to maintain the speed for your car
- **“Stop boxes”**
    - Again, the two stop boxes are red paper on the ground
    - Brainstorm and do research to find a good solution that involves processing of the image to see the red area and accordingly stop the vehicle
        - If at the first stop box, continue moving after stopping
        - If at the second/final stop sign (end of the track), stop permanently
    - You will have to be conscious of how this check/processing affects the performance of your vehicle in terms of lanekeeping
        - Feel free to employ tricks here, e.g. we check for stop boxes every 3rd frame
- **Stop sign [graduate teams]**
    - If you are a graduate team, then this is your extra requirement
    - The stop sign will be a printed stop sign on paper that you can attach to any surface, such as a box, to keep it at a reasonable height/distance/location
    - Your car must use machine learning (e.g., YOLO or something similar) to identify the stop sign and take action
    - You can have the stop sign anywhere along the plane that is perpendicular to the ground and intersects the axis formed by the two yellow tape strips that say “TRAFFIC LIGHT”.

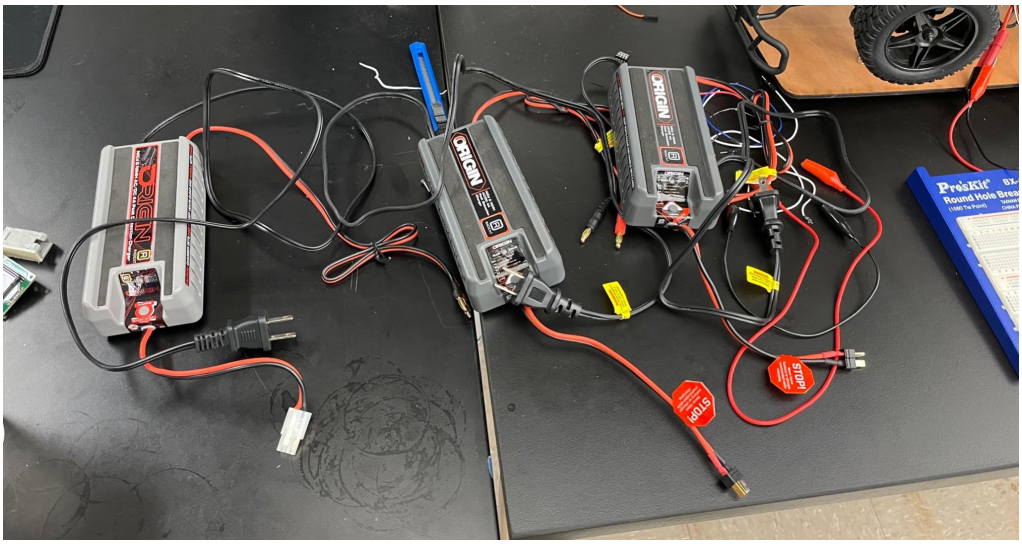


        - This means that you can place the stop sign at any height above the TRAFFIC LIGHT “line” that passes through the two yellow strips
        - If this doesn’t make sense, let me know and I can explain more.
    - Similar to the stop box, feel free to employ tricks here (e.g. only check every 3rd frame, don’t look for a stop sign after you find this one, etc.) to keep processing fast for the primary task of lane keeping
- **Battery charging**
    - There are two batteries to charge: The 7.2V battery used for car steering+speed and the portable chargers (5V/3A) used for the BBAI64
    - I recommend frequently charging the 7.2V batteries for car performance
        - See the following picture of the 7.2V batteries. Only use these yellow ones, I believe the other ones are garbage, but I could be wrong.



        - You can usually tell that things are slowing down or no longer working (e.g. no second beep for ESC initialization) when the battery is low

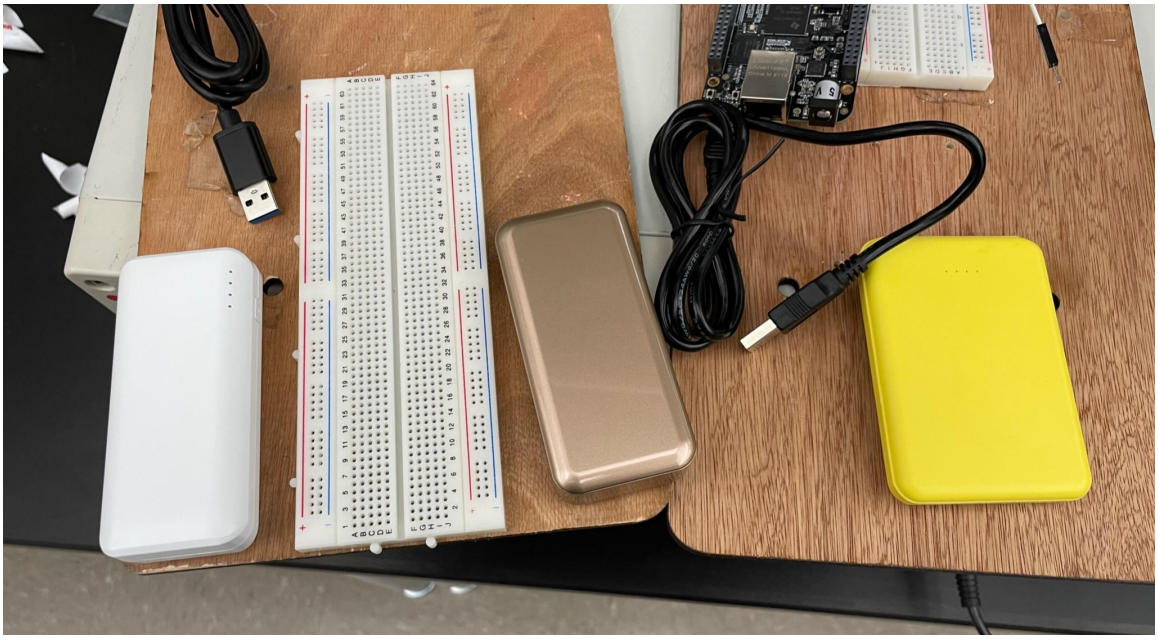
- To charge it, use any of these chargers in the picture. The darker two chargers now have adapters to easily connect with the batteries.



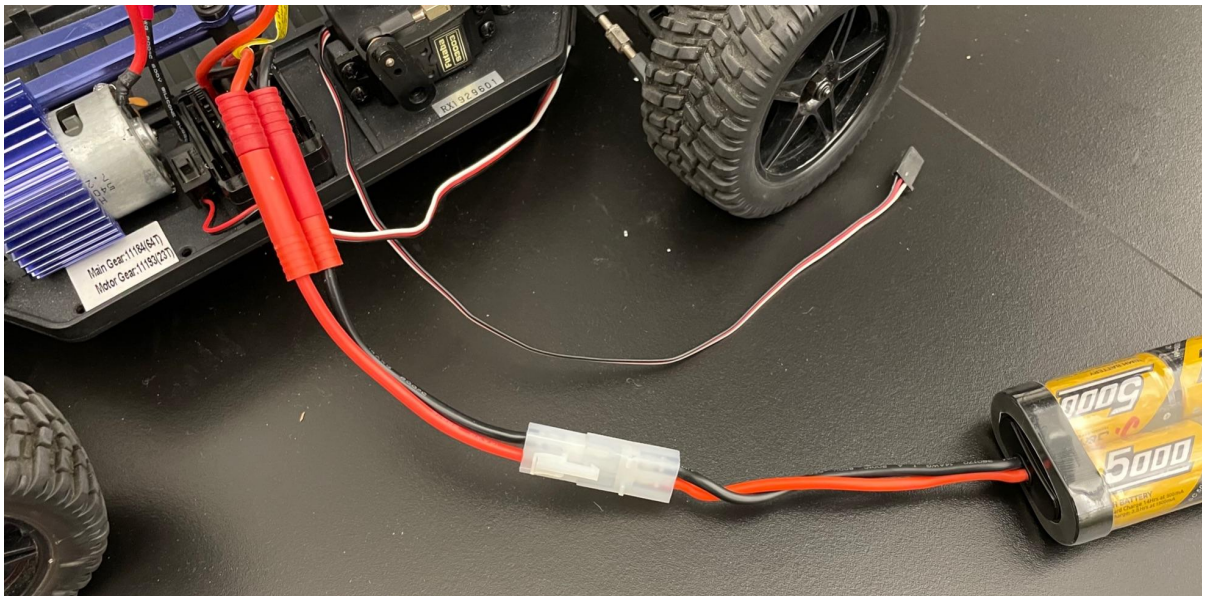
- For the charger, you plug it into the wall, then you'll see a slowly blinking red light.
    - Press the button a few times until you see a repeating sequence of 4 red flashes. This indicates that the charger is ready to flash at 4A, which is the fastest charging that it supports.
    - Then connect the battery to the charger, and hold down the button. You can let go when you see a permanent red light. This means the battery is charging.
    - It will eventually flash green when it is close to done, and show constant green when the battery is fully charged.
- **PLEASE ONLY HAVE CHARGERS PLUGGED INTO THE WALL AND BATTERIES CONNECTED TO THEM WHEN YOU ARE AROUND. IF YOU CONNECT A CHARGER TO THE WALL AND/OR A BATTERY TO THE CHARGER, PLEASE DISCONNECT THE CHARGER FROM THE WALL AND THE BATTERY FROM THE CHARGER IF YOU ARE LEAVING THE ROOM. OTHERWISE THERE IS A FIRE RISK.**

- The portable chargers (older ones in the following photo) for the BBAI64 can be charged using USB ports on your laptop or on the black power ports sitting on

the desks of the soldering area of the lab.

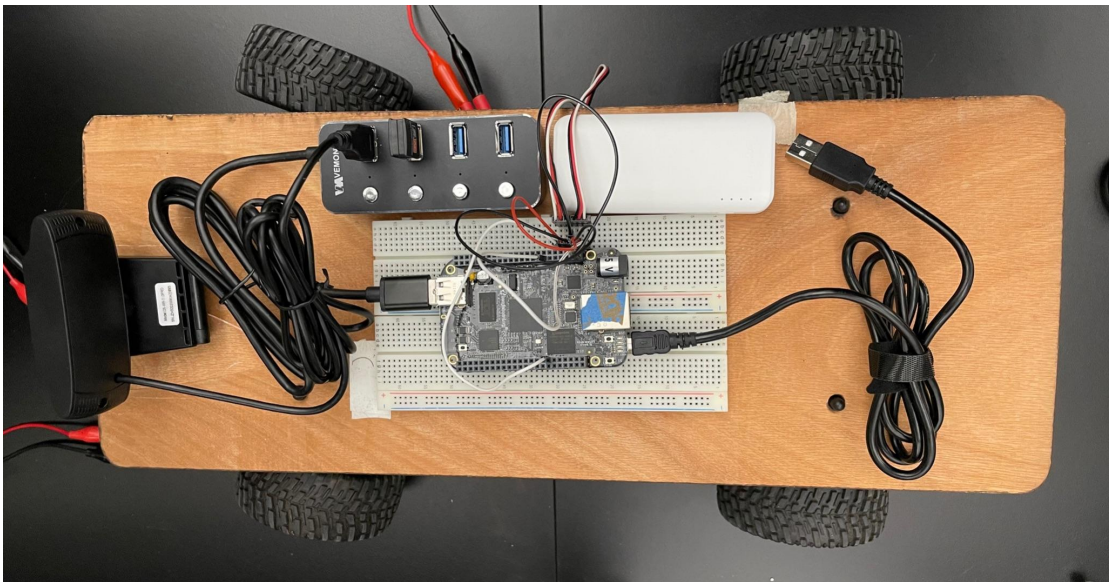


- **Connecting the battery to the car**
    - Connecting the 7.2V (yellow) battery to the ESC/servo of the car is as simple as shown:



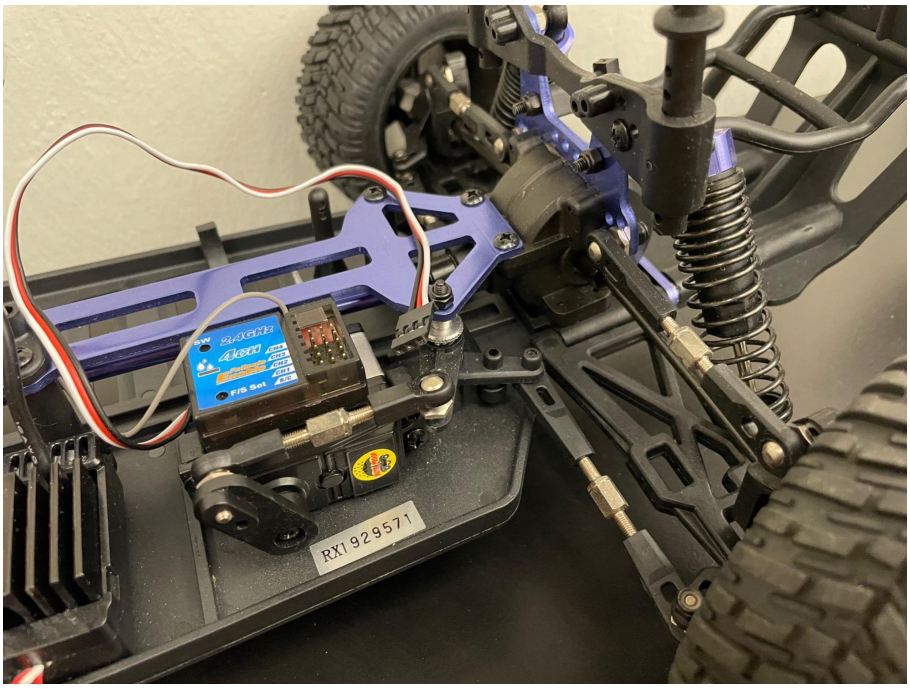
    - **PLEASE DISCONNECT THE 7.2V BATTERY FROM THE CAR IF YOU ARE LEAVING THE ROOM. PROBABLY NOT A FIRE HAZARD BUT I AM STILL USING ALL CAPS**
- **Connecting your BBAI64 to the car**
    - Please use a single wooden board (see older pictures below; these are slightly different from this year, since for example you will not use a USB hub) for your team and use blue/yellow tape with a sharpie to affix your team name to the

board. There are a bunch in the lab. There is also a box of jumper cables in the lab.

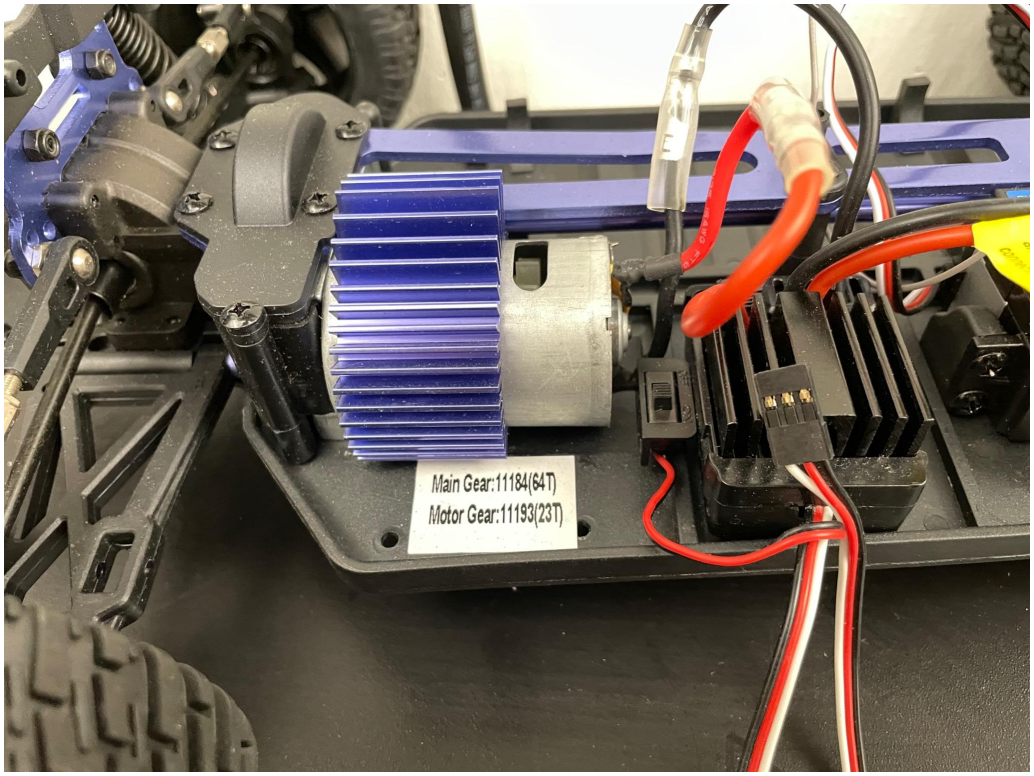


- The beauty of these boards is that it makes it so you can have all of your equipment on the board. Then, you just place the board on any vehicle, connect some jumper cables from the car, and then run your code. Once you're done, you can disconnect jumper cables and remove the board from the vehicle.
    - Unfortunately we have to share vehicles - there are currently six that work.
- Feel free to use the velcro in the lab to affix your components to your board.
- For the connections between the car and the BBAI64 (be very careful - do not let jumper wires get near any pins that you don't intend to connect them to; DO NOT CONNECT GROUND TO A PWM PIN):
    - Your BBAI64 should have three connections to the car:
        - Common ground: BBAI64 ground, the black jumper of the ESC, and the black jumper of the steering servo should all be connected
            - There are some new cars where the servo wires are yellow, red, brown instead of white, red, black. Since I say the latter colors in my instructions, for those cars (for the servo) you should treat yellow as white, red as red (nice), and brown as black
        - One BBAI64 PWM pin, e.g. P9_16 (you can figure this out by reading the labels on the board for P9 and the numbers; **NOTE: P9 on the BBAI64 has 4 extra pins at the top labeled with E that precede the usual P9_01 and so on; P8 doesn't have this**), should be connected to the white jumper of the steering servo

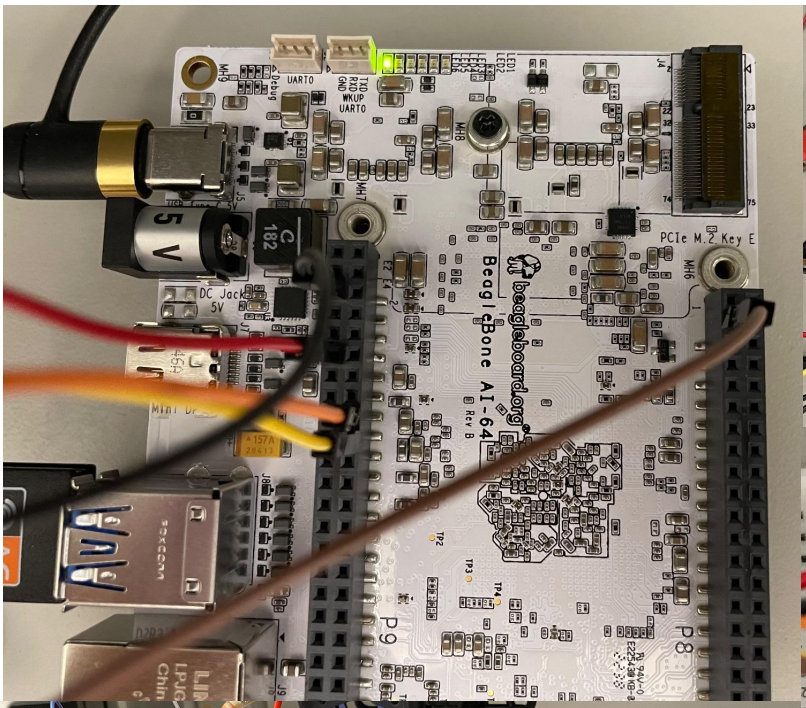
        - The steering servo is connected to the two front wheels - see the following picture



    - Another BBAI64 PWM pin, e.g. P9_14, should be connected to the white jumper of the ESC
        - The ESC is connected to the big motor - see the following picture



- The red jumpers of the ESC and steering servo need to be connected, but DO NOT connect the red jumpers of the ESC and servo to the beaglebone
    - ESC and servo red jumpers are connected to give the steering servo power
- I recommend making your own kind of cable that can attach to these things, when you are in the lab look at how I have made a custom “connector” for the cars. You can use tape to make a “connector” that interfaces with this. You can use green tape to help remember the polarity of the connector.
- The following are pictures of my setup, which closes the instructions.


BeagleBone AI-64
beagleboard.org
P9
P8
PCIe M.2 Key E
DC Jack
5V