ELEC 424 - Project 3: Motor Control

100 points

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. First you'll get some motor control implemented for project 3. The remaining tasks for motor feedback and lane keeping will be the final project.

To make motor control happen for project 3, 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]. 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 AI-64 (BBAI64) attached to the RC cars in Ryon. Your job for project 3 is to use a Raspberry Pi (undergrad) or a BBAI64 (graduate) to provide speed and steering control of the RC car.

This is a team submission. Each team should have 1 demo and 1 submission for Canvas.

NOTE for graduate teams: BBAI64 has 4 extra pins at the top on one side that have "E" labels (see first picture here). Then, the normal indexing (normal for BeagleBone devices) of P9_01, P9_02, etc., occurs. Take this into account! P8 is the same between the BBAI64 and BeagleBone Black. 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. **(50 points) In person or submitted recorded video demonstration (here) of functionality performed by a single Python script (the script should do all of the following). In person demonstrations can be done during instructor or TA office hours. All demonstrations should be done with having the car on top of something so the car itself doesn't move, but the motors/wheels do.**

a. (10 points) Hardware interfacing between Linux device and motor control set up correctly
b. (10 points) Python script control of speed motor demonstrated (can make car wheels spin forwards)
c. (10 points) Python script control of stopping speed motor demonstrated (can make car wheels stop spinning)
d. (10 points) Python script control of servo motor demonstrated (can make car turn left and right)
e. (10 points) Python script control of straightening servo motor demonstrated (can make car wheels turn straight)

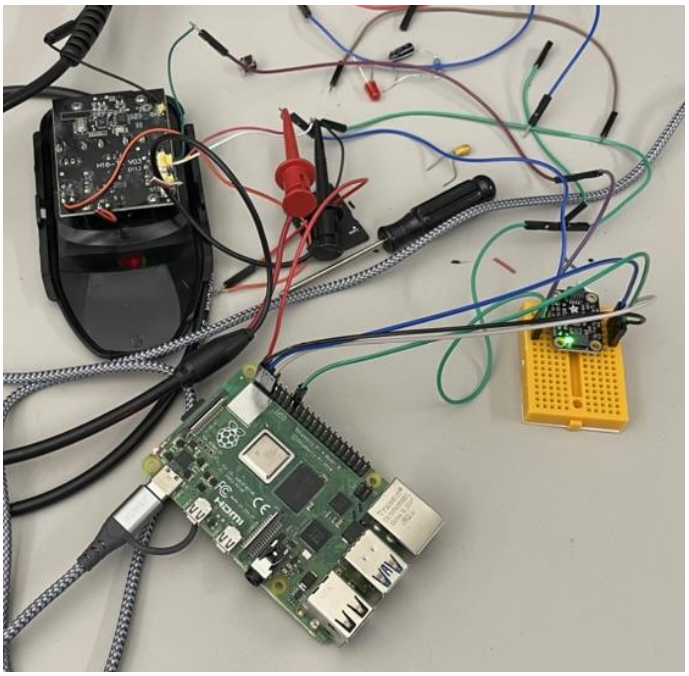
2. **(20 points) Submission of relevant commented code file(s) to Canvas**
    a. (10 points) Code attempts to achieve objectives/requirements stated in instructions
    b. (10 points) Code reasonably includes comments (at least every other line includes a comment)
    c. (5 points) The following file(s) must be submitted in source form (.tbl, .c, .py, etc.) - not a PDF
        i. Your python script that executes point 1 of this rubric
        ii. You must cite the following Instructable at the top the file as inspiration:
            - 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. **(5 points) Submission of MAC address of wireless interface submitted to Canvas**
    a. Upload a text file that has the entire ifconfig output from the terminal ssh'd into your Linux device
4. **(25 points) Submission of PDF report to Canvas**
    a. (1 point) Title of assignment/project
    b. (1 point) Your team member names
    c. (4 points) 1 paragraph (at least 4 sentences) describing the goal of the project and the steps you took to complete it (include a statement on each key function)
    d. (4 points) A 2nd paragraph (at least 4 sentences) describing what you found challenging/any bugs you had to fix, what you learned, and what you think would be another interesting application for this approach.
    e. (4 points) Include a screenshot showing a significant portion or all of your code.
    f. (4 points) Include a picture of your hardware setup, including the car and the Linux device

g. (4 points) All screenshots in the report must include a figure label with a short description.
h. (3 points) You must cite that this work draws from the following Instructable:
    i. 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/
    ii. Also be sure to cite any existing 424 projects on that site that you used

Guidelines

- **Back up your code frequently!**
    - Your BeagleBone AI-64 or RPi could get fried
    - If you break it, you will have to make the project work on a different platform possibly.
- **Linux devices**
    - Raspberry Pi (undergraduate)
        - You should flash a microSD card just like you did at the beginning of the semester and use this for the Pi
            - Have one teammate use one of their cards
        - Here are my pro tips for reliably flashing the SD card:
            - Flash the SD card using Etcher with this specific image here
            - Once the SD card is flashed, remove and reinsert it into your computer (don't put it in the Pi yet!)
            - Download and drag the files from here onto the root of the SD card (which will overwrite some files)
            - Put the SD card into the Pi and plug the Pi in to your computer using the USB-C port of the Pi
            - Wait 120 seconds, then try the ssh command using a terminal (ssh pi@raspberrypi.local)
            - Run the command “sudo raspi-config” and go through Network Options to set up Rice Visitor
            - WiFi should then work (reboot and try again if it does not)
    - BeagleBone AI-64 (graduate)
        - You should be able to SSH into these boards using a USB-C cable (they are already flashed, no SD card is needed; they have built in storage)
            - WiFi should already work for Rice Visitor
        - The SSH command: ssh debian@192.168.7.2
            - On some platforms 7.2 should be replaced with 7.1, 6.2, or 6.1

        - NOTE: Typically an Apple Silicon Mac and possibly other laptops cannot power the BBAI64 reliably; You may have to use another laptop or any of the desktops on Campus (they all have command prompt which supports ssh)
            - You will notice this if lights are flashing and changing but you are not able to log in after 2 minutes
            - In this case, you won't be able to access the BBAI64 from some laptops until you give me the MAC address info and I ask IT to give a fixed IP address to your BBAI64 so you can log in wireless over Rice Visitor
                - Feel free to immediately email the MAC address file that is mentioned in these instructions so you can get this access sooner
- **MAC Address**
    - For the final project you will need to be able to access your Linux device remotely (over Rice Visitor)
    - Get the MAC address of the wireless interface of your device so I can request from it from IT
    - Steps:
        - SSH into your Linux device
        - Run ifconfig and save the entire output into a file that you upload to Canvas (email me the file immediately if you want access sooner)
- **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
- **Undergraduate teams only: Digital to analog conversion (DAC) and control**
    - You will open the hand controller for the RC car and cut the wires in the middle for the two potentiometers that control speed and steering
        - See picture for my setup
        - You only need to connect the middle wires of the two sets of three wires that were connected to the potentiometers (in my picture I connected more but not needed)
        - These will be connected to output of your DAC



- You will power the hand controller with the RPi using 5V and ground pins of the Pi
  - Cut the hand controller battery wires in the middle and connect them to the corresponding pins (red wire should be 5V, black wire should be ground)
- You will imitate a potentiometer using the given Adafruit DAC here
  - You will have to solder the headers to the DAC, use the same soldering setup in FEP102 as before
- Be sure to set the speed maximum to 30% on your hand controller - see the instructions for the car to do this
- On the software side, you need to install some python items for the DAC
  - As per Adafruit's guide here, run the following commands (I've modified the instructions somewhat):
    - sudo apt-get update
    - sudo apt-get install python3-pip
    - sudo apt install --upgrade python3-setuptools
    - cd ~
    - pip3 install --upgrade adafruit-python-shell
    - wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/raspi-blinka.py
    - sudo python3 raspi-blinka.py
    - (If anything is mentioned about reboot, say yes / reboot)
  - If those commands don't work, try the manual route (as per Adafruit's guide here)
    - sudo raspi-config nonint do_i2c 0
    - sudo raspi-config nonint do_spi 0
    - sudo raspi-config nonint do_serial_hw 0
    - sudo raspi-config nonint do_ssh 0
    - sudo raspi-config nonint do_camera 0
    - sudo raspi-config nonint disable_raspi_config_at_boot 0
    - sudo apt-get install -y i2c-tools libgpiod-dev python3-libgpiod
    - pip3 install --upgrade RPi.GPIO
    - pip3 install --upgrade adafruit-blinka
    - ls /dev/i2c* /dev/spi*
      - (output expected to be: /dev/i2c-1 /dev/spidev0.0 /dev/spidev0.1)
  - Once installation is done, follow the Blinka Test part of the guide from Adafruit here and verify that you get the same output

        - Now you need DAC code - follow the instructions (including the library install that is mentioned) here
            - You will set the voltages of the middle wires that were previously connected to the potentiometers to make the RC car move and turn
            - In my experience, 2.1 V or so was a neutral voltage that stopped the car
            - You will have to play with this to figure out voltage ranges
        - When working with the car, be sure to have the battery charged often (USB charger is included in the box)
            - I also recommend removing/unscrewing the top cover of the car
        - In my experience, if you cannot get the car to reset to not moving from your terminal, you will have to unplug the battery in the car
        - ALWAYS have the car propped up on something for this project so that it doesn't actually move

The remainder of this document is for graduate teams

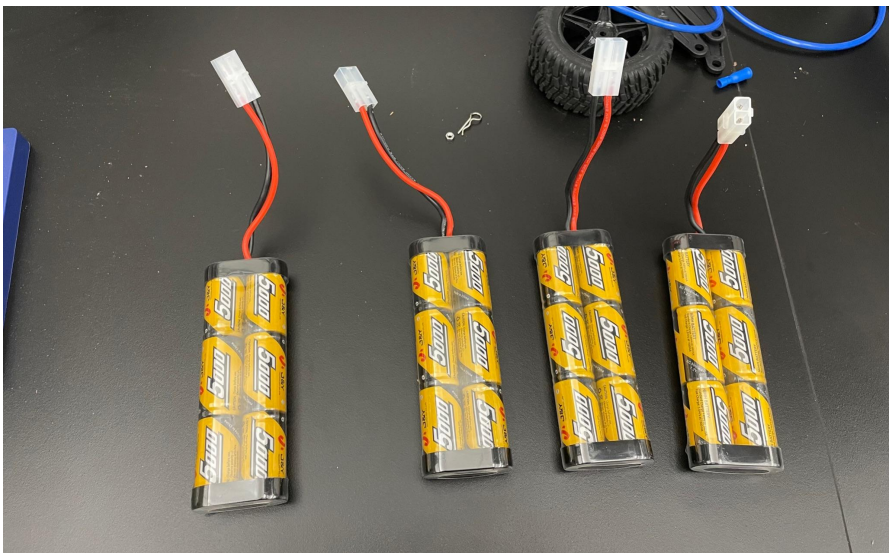
- **Graduate teams only: Pulse-width modulation (PWM)**
    - Many existing Python files use either an Adafruit PWM BBB library or a Raspberry Pi PWM library
        - These libraries don't work with the BBAI64 - aren't you feeling lucky? ;)
    - For the BBAI64 (based on the instructions here from user RobertCNelson on beagleboard.org), do the following commands to get PWM available:
        - ls /opt/source/
            - You should see a folder like "dtb-5.10-ti-arm64" but possibly with a higher number, let's call that folder YYYY
            - Email me with a screenshot of the terminal output if you see more than that one folder with something like dtb-y.yy-ti-arm64 and pause on the project before hearing back from me
        - cd /opt/source/YYYY/
            - Replace YYYY with the folder name mentioned before
        - 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
    - **NOTE: DO NOT RUN “sudo apt upgrade”, ever (for this project or the final project)**
        - This could break wifi and the device tree. You can still run "sudo apt update" and do "sudo apt install x" (x being whatever you are trying to install).
        - If you do run "sudo apt upgrade" [but again, don't!], then you would have to download the headers on your laptop, transfer them to your BB-AI-64, use dpkg to install the headers, uninstall the wifi driver, reinstall it using the new headers, and run the steps (e.g., cd /opt/source/dtb-5.10-ti-arm64/, git pull, etc.) for PWM again (making sure to not overwrite your PWM0.dts file when doing git pull). You can see what we want to save you from.
        - The reason this could break things is because there may be a kernel update. That means the headers (possibly) change, and numbers in folders' names change, leading to chaos.
    - 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):

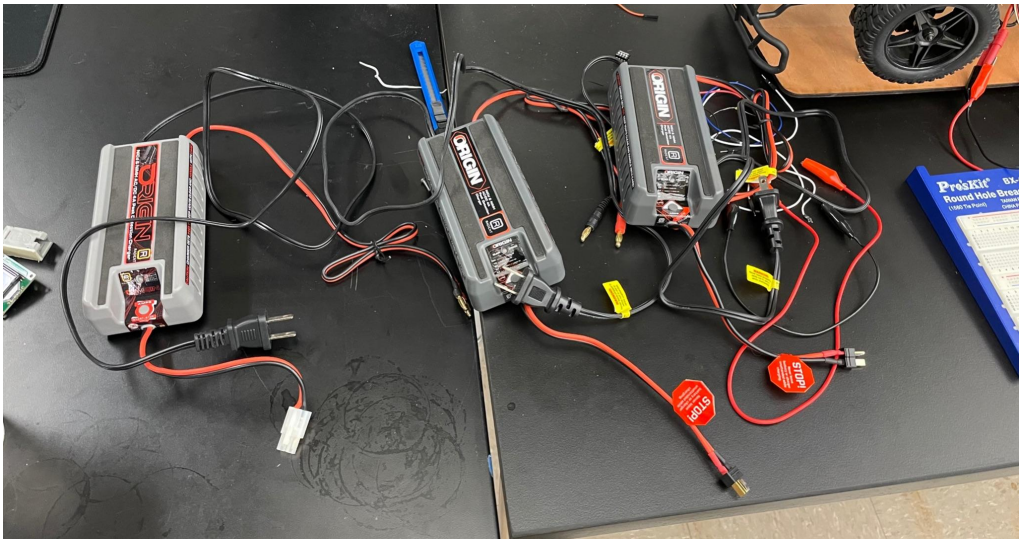
```
# P9_14 - Speed/ESC
with open('/dev/bone/pwm/1/a/period', 'w') as filetowrite:
    filetowrite.write('20000000')
with open('/dev/bone/pwm/1/a/duty_cycle', 'w') as filetowrite:
    filetowrite.write('1550000')
with open('/dev/bone/pwm/1/a/enable', 'w') as filetowrite:
    filetowrite.write('1')

# 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
    - **Always test the car on a propped up surface so it can't actually move**
        - **As a backup for the car running out of control, 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. You could also turn the switch off the ESC.**
    - 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 increase in careful steps for the ESC - the car will really take off if the duty cycle is high enough and likely break something when it hits a wall
            - I'd recommend doing tests for the car you are working with
            - Try starting at 7.9% and increment in steps of 0.05% to see when it starts actually moving, then you can do finer steps in your speed control algorithm if necessary
    - 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, but our car is more advanced)

- **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 some 7.2V batteries. Only use the ones you were given with the car or 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 the chargers you were given or any of these chargers in the picture. The darker two chargers now have adapters to easily connect with the batteries.

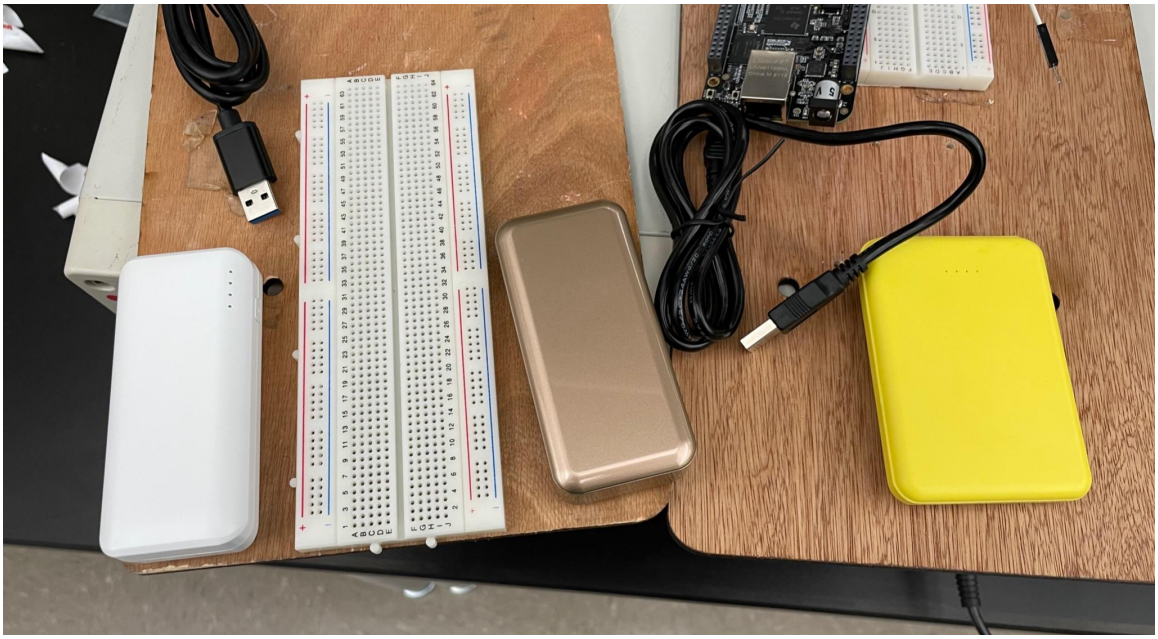


        - For the charger, you plug it into the wall, then you'll see a slowly blinking red light.

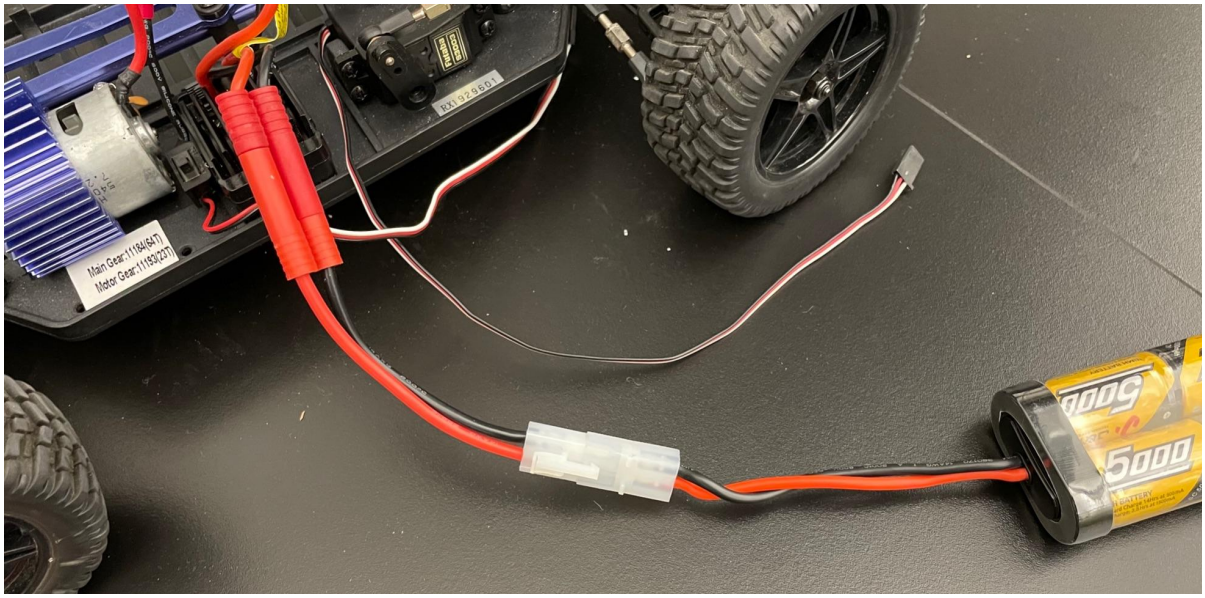
- Press the button a few times until you see a repeating sequence of 4 red flashes. This indicates that the charger is ready to flash at 4A, which is the fastest charging that it supports.
- Then connect the battery to the charger, and hold down the button. You can let go when you see a permanent red 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.

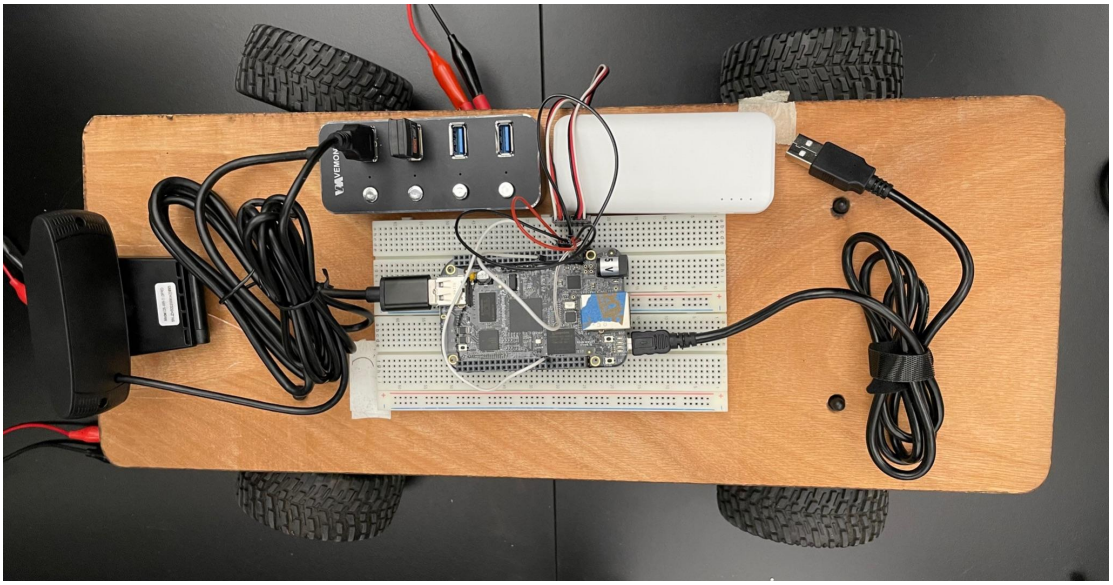


- **Connecting the battery to the car**
  - Connecting the 7.2V (yellow) battery to the ESC/servo of the car is as simple as shown:



- **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**

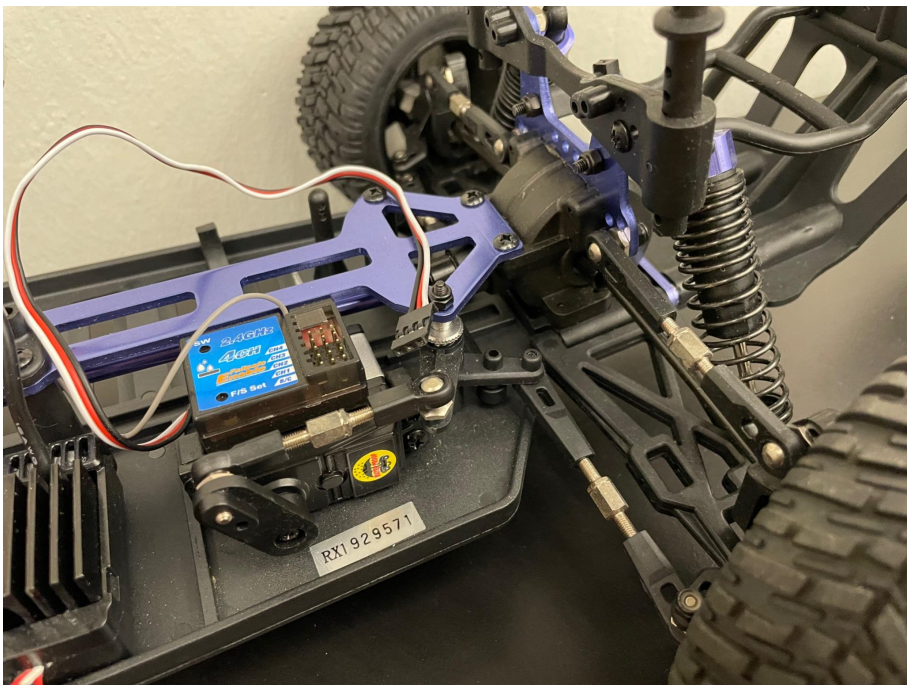
- **Connecting your BBAI64 to the car**
  - Please use a single wooden board (see older pictures below; these are slightly different from this year, since for example you will not use a USB hub) for your team and use blue/yellow tape with a sharpie to affix your team name to the board. There are a bunch in the lab. There is also a box of jumper cables in the lab.



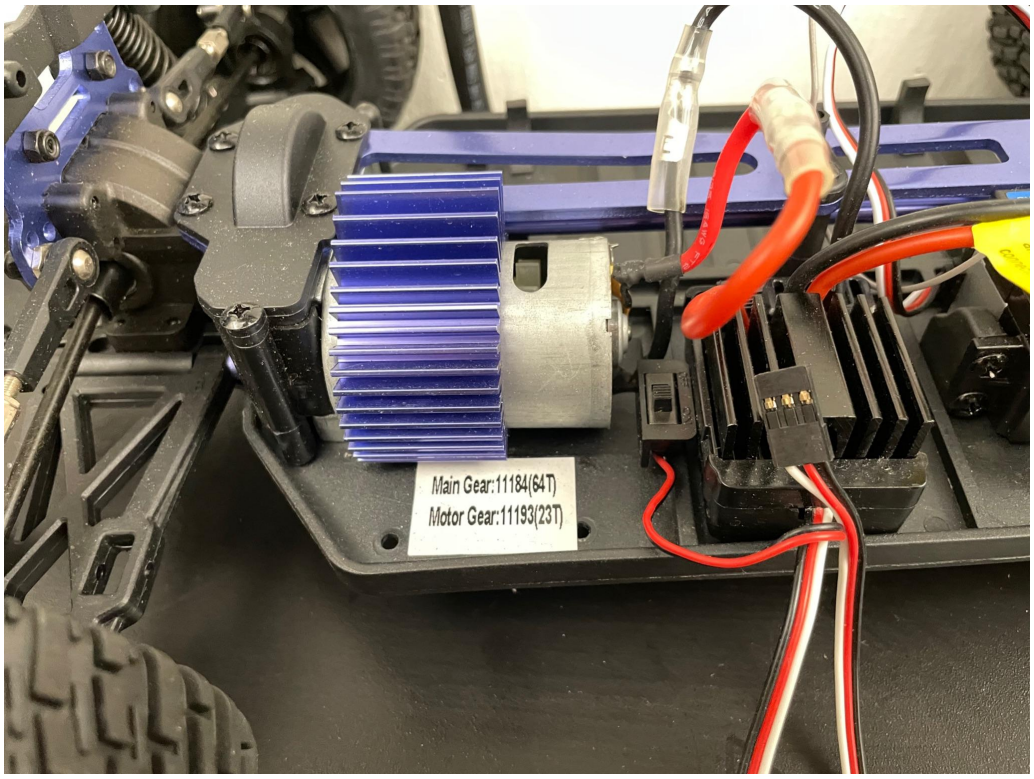
  - The beauty of these boards is that it makes it so you can have all of your equipment on the board. Then, you just place the board on the vehicle, connect some jumper cables from the car, and then run your code. Once you're done,

you can disconnect jumper cables and remove the board from the vehicle if you want.

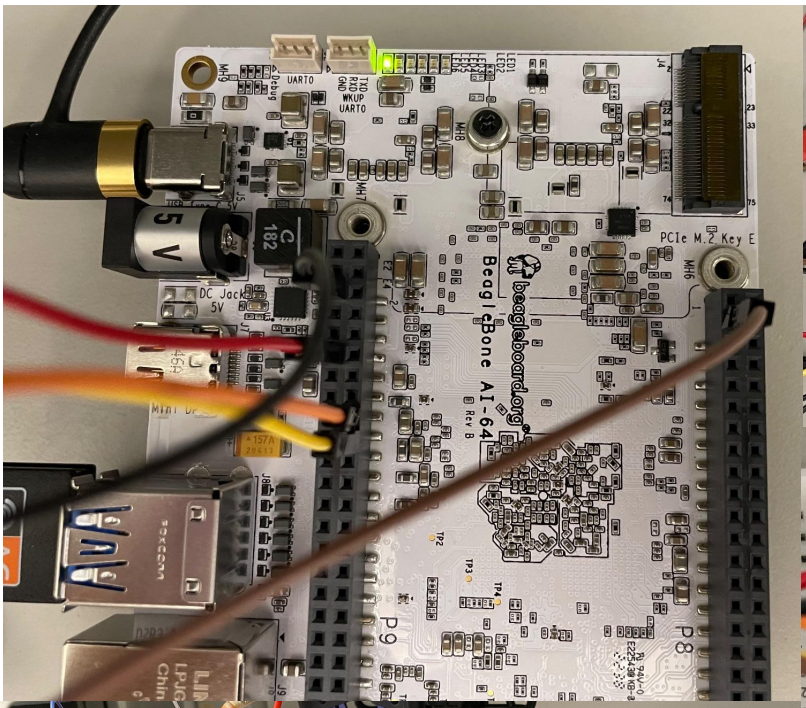
- Feel free to use the velcro in the lab to affix your components to your board. It is on the table with all the unlabeled cars.
- For the connections between the car and the BBAI64 (be very careful - do not let jumper wires get near any pins that you don't intend to connect them to; DO NOT CONNECT GROUND 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 other colors of 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