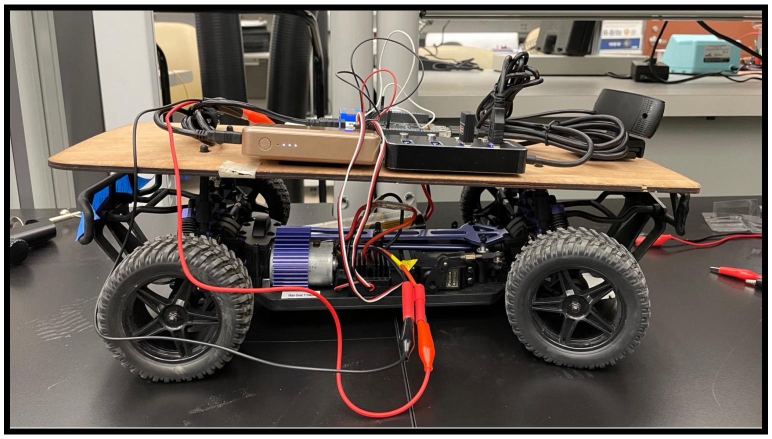
ELEC 424 - Final Project: Self-Driving Car

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.

You will work as a team to program 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 project 3 and previous years 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 Black (BBB) or BeagleBone AI-64 (BBAI64) attached to the RC cars in Ryon. Your job is to make this system work on a Raspberry Pi 4 (RPi 4) [undergraduate] or BBAI64 [graduate]. You will not only enable the car to perform lane keeping, but also to stop at an orange stop box on the ground and (if you are a graduate team) to identify drivable space using YOLOP. Grading will be based primarily on the performance of your car on any of the tracks laid out in Ryon B12, Ryon B10, or FE&P 102.

NOTE for graduate teams: BBAI64 has 4 extra pins at the top on one side that have "E" labels (see first picture here). Then, normal indexing (normal for BeagleBone devices) of P9_01, P9_02, etc., occurs. P8 is the same between the BBAI64 and BeagleBone Black.

Rubric

1. **(120 points) In person or submitted recorded video demonstration (here) of self-driving functionality. In person demonstrations can be done during instructor or TA office hours.**
    a. **(40 points) [30 points if graduate team] <u>Speed encoding</u>**

        i. (20/15 pts) System uses speed encoder to maintain a consistent speed
        ii. (20/15 pts) System implements driver to precisely measure timings between optical encodings and these timings are used to adjust the speed control

   b. **(40 points) [30 points if graduate team] <u>Lane keeping</u>**
      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 6 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] <u>Orange stop box on ground</u>**
      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. (10 points) Run YOLOP successfully on BBAI64 using webcam with video output displayed on laptop
      ii. (10 points) Estimate frame rate of YOLOP on BBAI64
      iii. (10 points) Record video output of YOLOP for webcam sitting in front of test track

      Note: No need to drive using YOLOP; you are just testing YOLOP's algorithmic performance on the BBAI64

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 5 components (webcam, RPi or BBAI64, USB WiFi adapter (if grad team), optical speed encoder, 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 YOLOP.
    iv. (4 points) On one plot show error, steering voltage or duty cycle, and speed voltage or duty cycle 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 the vehicle with your hardware on it
    vii. (4 points) Video of the vehicle completing the course with your hardware and software running on the vehicle

  g. (4 points) Code
    i. Please upload any .py files, C files (including driver code for optical encoder), and/or other code used for the project
        - Graduate teams: You must include your modified device tree source file (the PWM 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

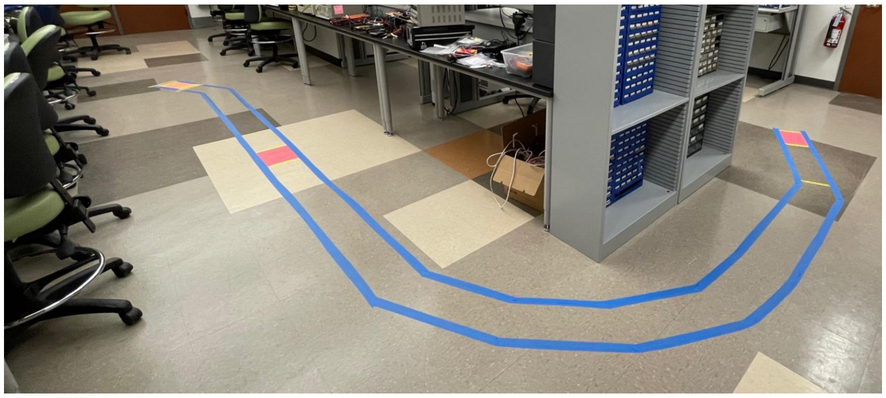
3. **(20 points) Evaluation of teammates’ effort**
    a. (5 points) Fill out a form estimating teammates’ effort
    b. (15 points) The average of teammates’ estimates of your effort

Note: The evaluation will be about whether or not everyone did at least their fair effort.

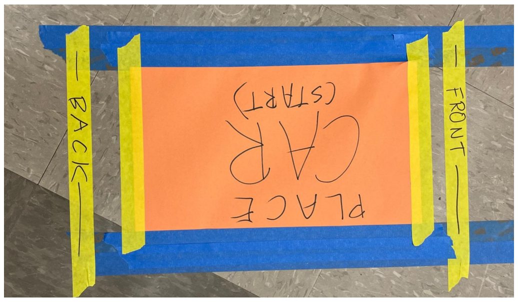
4. **(20 points) Submission of relevant commented source code files to Canvas**
    a. Code attempts to achieve objectives/requirements stated in instructions
    b. Code reasonably includes comments
    c. The following file(s) must be submitted in their source form (e.g., .py, .c, .dts) - not a PDF. Only 1 team member needs to submit the code.
        i. Any .py file(s) and/or other code used for the project
        ii. Graduate teams: You must include your modified device tree source file (the PWM 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

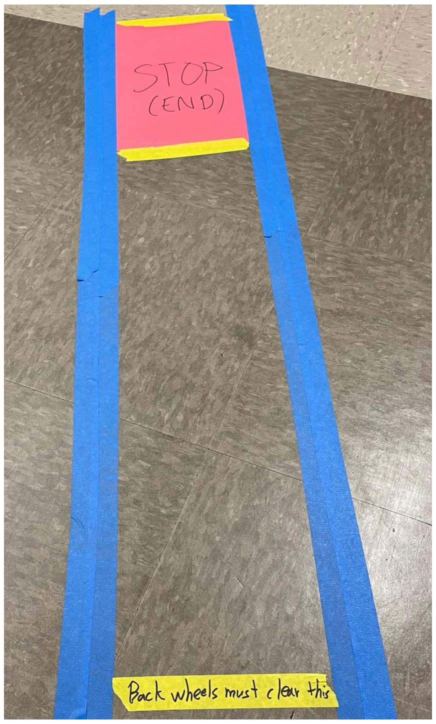
- **General**
    - Back up your code frequently! Your RPi 4 or BeagleBone AI-64 could get fried. If you break it, you may have to make the project work on another platform, and this will not be fun!
- **Track**
    - The following is a picture of one of the tracks. The start is on the left, and the end is on the right. Your car must successfully navigate the track, stopping at the stop boxes (red [now orange] paper on the ground). The car should completely stop at the second/final stop box.



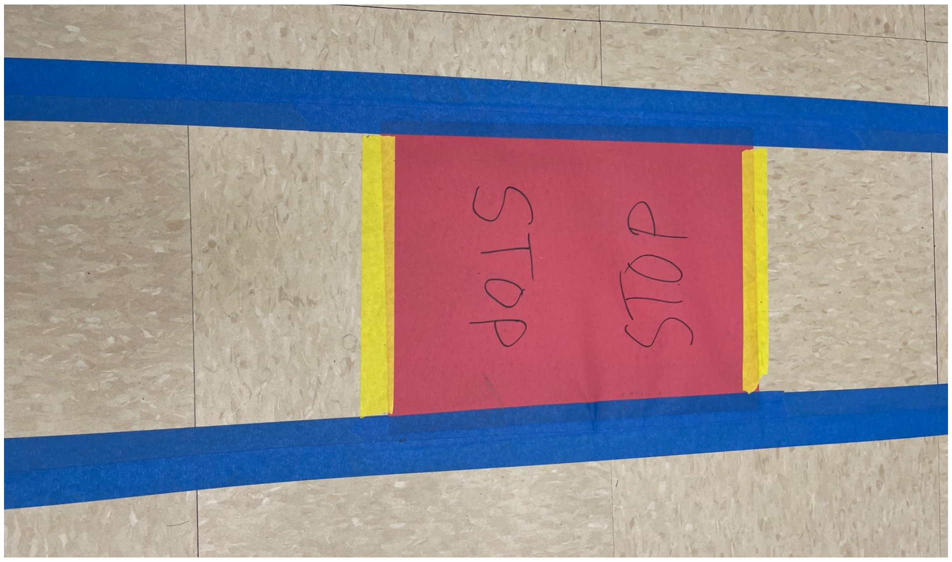
    - The following picture shows the start where the car must be placed



- ○ 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 (now orange) piece of paper on the ground :)
- ○ Similarly, there is a red (now orange) stop box in the middle of the track as well



- ○ The rest of these guidelines detail requirements and advice around speed control through optical encoding, lane keeping, and the stop boxes

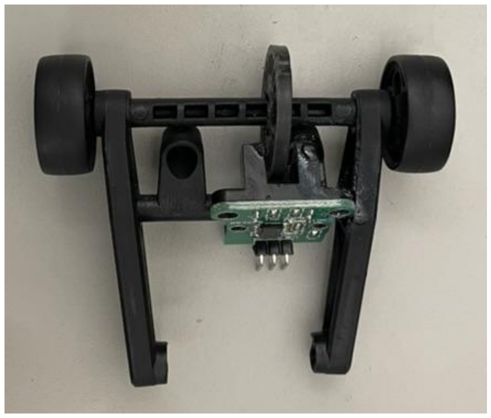
- **Code to start with**
  - ○ Start with one of the 424 project files here and/or the lovely Python file provided on Instructables (Autodesk, Inc.) here by user raja_961 in combination with your project 3 code

- Note that the Instructables code just turns the car left or right in a binary fashion
- For our course it is required that you perform “analog” steering, not just full left or full right; This will be done via PID (see later section)
- 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 motor control) 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 back to neutral so the ESC stops moving the motors.**
- **As a backup, you can (1) have a Python function that sets motor outputs to neutral 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. Graduate teams: You could also turn the switch off the ESC.**

- **Lane keeping**
    - Once you’ve given me your MAC address info and I have asked IT to assign a fixed IP address to your Linux device, you will generally interact with the car by wirelessly ssh’ing into the Linux device attached to it (using the fixed IP address rather than raspberrypi.local or 192.168.7.2).
    - By doing so, you can wirelessly tell the car to start its Python lane keeping/self driving script. If you ssh in with X or Y 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 (downsize) 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
    - Images/video window not showing? You may need to add `cv2.waitKey(1)` in your core loop if it is not there already
        - More info - MLK, URL: https://machinelearningknowledge.ai/opencv-cv2-waitkey-tutorial-with-examples/
- **PID/PD controller for steering**
    - The project programs on Hackster implement 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
  - Undergraduate teams: You will have to test to figure out the voltage range corresponding to full left and right turns, so you can make small turns for small error and large turns for large error
  - Graduate teams: 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/or 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**
  - You have been given an optical encoder like this one
  - **Undergraduate teams**: Mounting the encoder setup will be a creative exercise for you - see my picture for how I superglued (there is super glue in Ryon b12) the encoder to the frame for the extra back wheels included in your kit. This can be attached to the back of your car, enabling the encoder wheel to spin on the ground as your car moves. You may have to add more weight to this frame for it to work properly. You also will have to drill through the encoder wheel to be able to put the rod through it so it can spin around the rod. If you cannot find access to a drill, we should have a drill in the lab by Wednesday Nov 15.



  - **Graduate teams**: I was nice - your encoder setups should already be mounted, just test to make sure they work.
  - 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 Linux device
            - GND - connect this to ground
            - OUT - connect this to a GPIO pin of your RPi 4 or pin P8_03 of your BBAI64
    - You can modify the project 2 driver to count the time between optical encoder activations, since the optical encoder will act like a button
    - **<u>Graduate teams Only</u>**: 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
        - Replacing Y's with what exists in the folder, you want to add the following two lines starting at line 10 of the /opt/source/dtb-Y.YY-ti-arm64/src/arm64/overlays/BONE-PWM0.dts file:

          ```
          #include <dt-bindings/gpio/gpio.h>
          #include <dt-bindings/board/k3-j721e-bone-pins.h>
          ```
        - Then add this to the end of the same file (changing hello to whatever is in your driver file):

          ```
          &{/} {
                  gpio-leds {
                          compatible = "hello";
                          userbutton-gpios = <gpio_P8_03 GPIO_ACTIVE_HIGH>;
                  };
          };
          ```
        - Then you will follow a similar process to the PWM enabling you did for project 3 (again replacing Y's):
            - cd /opt/source/dtb-Y.YY-ti-arm64/
            - make
            - sudo make install
            - sudo reboot
    - 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
        - You can also grab capacitors from the shelves in Ryon B12 to put between ground and the OUT output of the encoder, since this will help "debounce"; Try different capacitance values.
        - I also changed the gpiod_set_debounce second input argument to 1000000 for time in microseconds. Seems wrong, but works well enough for me. Not sure why!
    - Somewhere in your main python script that is handling the system, you will want to periodically find out the rate of encoder wheel turning that your button/encoder driver is recording

        - You could make this a module parameter and read it in sysfs, then adjust motor control accordingly to maintain constant speed
- **“Stop boxes”**
    - Again, the two stop boxes are orange paper on the ground
    - Brainstorm/do research/look at prior teams’ code to find a good solution that involves processing of the image to see the orange 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
- **Graduate teams only: YOLOP**
    - If you are a graduate team, then this is your extra requirement
    - You must install and run YOLOP on the BBAI64
    - You do not need to use the ML acceleration capabilities of the BBAI64
    - Estimate and record the frame rate of YOLOP running on your BBAI64
    - Record video output of YOLOP using the BBAI64 with a webcam sitting in front of test track and note whether or not it can detect the lane markers
    - No need to drive using YOLOP; you are just testing YOLOP’s algorithmic performance on the BBAI64