ECE 456 and ECE 556 Project Description

Contents

ECE 456 and ECE 556 Project Description

Project Tasks Description

Task 1: Path Tracking and Task 2: Parking

The objective of this task is to design and implement a PID controller for a Lego Mindstorms EV3 Unmanned Vehicle (UV) for path tracking and parking. Figure 1 shows an example of a path r with a block placed at the end of the path.

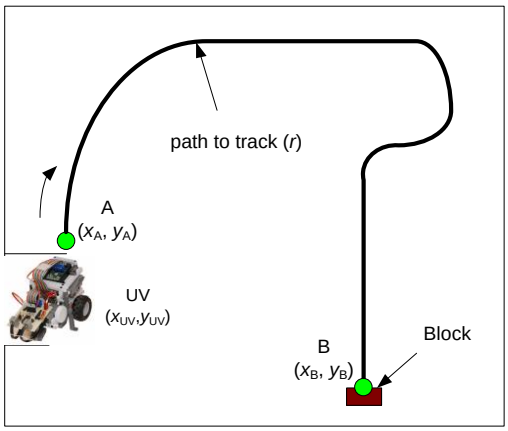


Figure 1. A project setting example.

The width of the path is around 2 inches (the one we are using is 1.88 inches bought from Amazon). The radius of curvature of the segments in the tracks will not be smaller than 0.5 ft. The path is designed using modules. The path set has 10 different blocks (the detailed file can be found in course website project section). During demonstration, 2 of the blocks will be chosen by the TAs to form the path for evaluation.

Each group will use the sensors, actuators, and control knowledge learned in the Mechatronics course to perform the following tasks:

1. **Path tracking** - Implement a PID control for the UV to move from point A with coordinate (x_A, y_A) to Point B (x_B, y_B) as fast as possible by tracking a given path r;
2. **Parking** - Park as close as possible to the block (Point B) without touching it;

FAQ for Task 1 and 2

- Q: Are we required to use all 3 light sensors? For example, can I build my robot with 3 light sensors and only actively use 2 of them in my program?
- A: You can use as many sensors as you want to control your EV. The TAs will use just the data from the anchor sensor to evaluate your tracking performance. The data from the anchor sensor will be used to calculate the mean square tracking error. The first derivative of the data will be used to determine the smoothness of your tracking algorithm.
- Q: Will each team have a different path to follow?
- A: No all teams will use the same path so that they will be evaluated with the same environment. The path used on the final demonstration will not be revealed until the day of the demonstration.
- Q: Is the board material (obstacle) the same at the track material (dry erase board from Lowes)?
- A: We are using a corrugate box as the obstacle.

Task 3: Platooning

The objective of this task is to implement a PID controller to regulate the UV speed to follow a moving UV in the front and maintain a constant distance. Your UV needs to follow the UV constructed by TAs moving at varying speeds (20% - 100%) in the front, as shown in Figure 2.

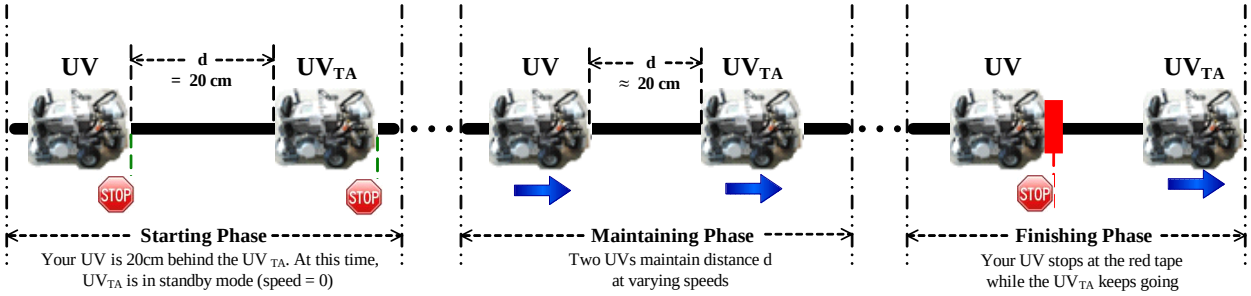


Figure 2. Platooning Task.

FAQ for Task 3

- Q: Does the TA robot have a white board affixed to the back? What is the size of that board?
- A: Yes, it is small, 0.75'x0.75' at most.
- Q: I plan to use a light sensor to detect the red tape. Is it safe to assume that there will be no other colored tapes on the path that could confuse the light sensor since it detects the light intensities only?
- A: There will not be any other color tapes on the path except for black and red.
- Q: How Can I test the performance of my platooning task?
- A: For On-Campus student, you can work with other teams to use their UV as the TA bot and test your platooning performance or use a cardboard and move it in front of your UV. For Distance Education students, you can use a cardboard and move it in front of your UV to test the platooning performance. We will not provide TA bot or TA bot code to you to test.

Building the UV

Overview

Each group will use the following components to build the UV:

- Lego Mindstorms EV3 Kit (all sections);
- Light sensors:
  - Lego light/color sensors (ECE 456 and ECE 556 distance education students)
  - Customized light sensors (ECE 556 on-campus students only);
- Ultrasonic sensor:
  - Lego ultrasonic sensor (ECE 556 distance education students only)
  - Customized ultrasonic sensor (ECE 456 and ECE 556 on-campus students);

Here are few typical UV designs & setups:

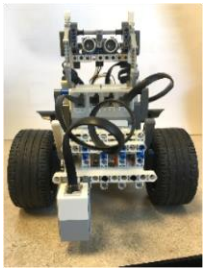
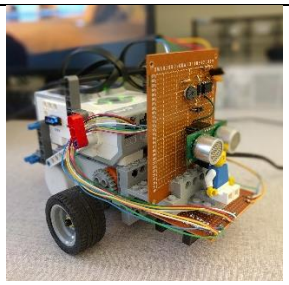
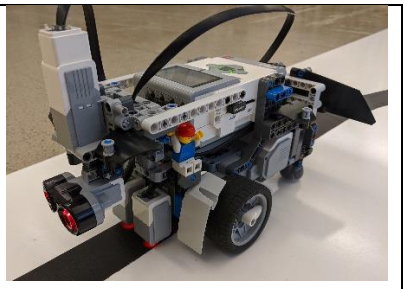
		
ECE 456	*ECE 556 on-campus*	*ECE 556 DE (Distrance education)*

Figure 3. Typical UV designs.

Building the Light Sensor Module

You will use up to three light sensors, in a configuration that you see fit, to detect the path and generate appropriate signals for the UV to track the path. The position and configuration of the light/color sensors can highly influence the performance of your UV. In this project, **one light/color sensor (anchor sensor) must be placed along/on the right edge of the track**. The **raw data** from the anchor sensor will be used to determine the tracking error (C2) and tracking smoothness (C3). The location of the other two sensors is up to you. Some examples are shown in Figure 4, and the anchor sensor position is highlighted in yellow.

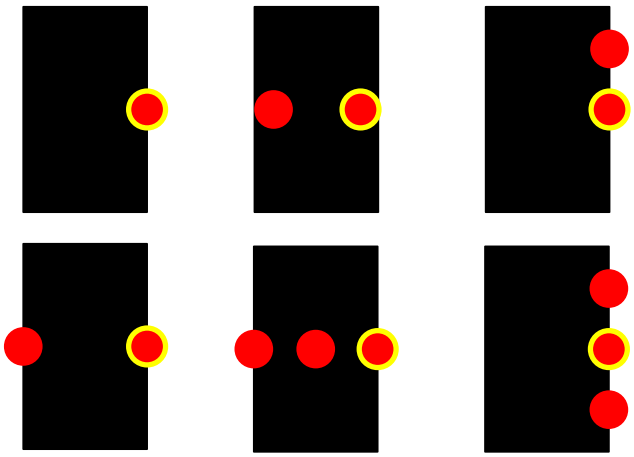
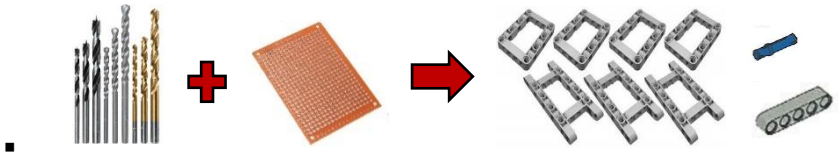


Figure 4. Typical light sensor configurations.

Tips for Building the Light Sensor Module

- Follow the Lab 4&5 manual to build the light sensor circuit.
- ECE 556 on-campus students: The EE-SF5(-B) photodiode datasheet recommends 5 mm sensor height.
- ECE 456 and ECE 556 DE students: The distance between the LEGO color sensors and the ground should be between 1~1.5cm.
- How to mount the circuit board on the LEGO UV?
    - Do NOT use glue!
    - Do NOT use tapes!
    - Find appropriate drill bits, and drill through-holes on the circuit board, then use lego components to mount and secure the light sensor assembly. (Available at Maker's Space) – *Sign-up for a training to get access*.



Building the Ultrasonic Sensor Module

You will use the ultrasonic sensor to detect the distance between the UV and the block at the end of the track for parking task. You are expected to park as close to the block as possible. ECE 456 and ECE 556 on-campus students are required to build your own ultrasonic sensor using the SRF04, whereas ECE 556 DE students can use the stock LEGO ultrasonic sensor.

Tips for Building the Ultrasonic Sensor Module

- ECE 456 and ECE 556 on-campus:
  - The example circuit for SRF05 is listed in Lab4&5 manual.
  - How to mount the circuit board on the LEGO UV?
    - Do NOT use glue!
    - Do NOT use tapes!
    - Find appropriate drill bits, and drill through-holes on the circuit board, then use lego components to mount and secure the sensor assembly. (Available at Maker's Space)
    - Do NOT solder the SRF05 sensor on the circuit board (we will recycle the SRF05 sensors)
      - Use the female pins to host the SRF05 sensor
    - 

Programming the UV

There are many other tools to program the EV3 (such as Lego Mindstorm, MATLAB/Simulink, ev3dev) and you are welcome to use whatever the tools you want to program your Brick. However, the TAs will not provide any technical supports other than Lego Mindstorm and Matlab/Simulink. For example, if you get stuck with ev3dev on the Wifi connections, you will be on your own. You can consult the related forums if they can help. The overall pros and cons for each platform is listed in the Figure 1 below. Some program examples can be found in the T06 slides.

Table 1. Programming platforms comparison.

	Minimum T_{sample}	Flexibility	Stand-alone mode?	Data acquisition	TA Support?	Applicability
MATLAB script	30~50ms via USB	Good	No	PC	Yes	All sections
MATLAB Simulink	4ms	Good	Yes	PC	Yes	ECE 556 DE
Lego Mindstorms	4ms	Okay	Yes	EV3	Yes	All sections
ev3dev	4ms	Great	Yes	EV3	No	All sections

Requirements for Programming the UV

1. The sampling time for all sensors should be smaller than 50 ms: $T_{sample} < 50$ ms.
2. Record the program runtime, anchor sensor, and ultrasonic sensor data for all tasks.

Project Evaluation

Your project will be graded according to your relative standings with respect to other groups in each section. Each group will have **three trials** for all the tasks. The TAs will select the best trial among your three trials for evaluation.

Task 1 and 2 will be evaluated in the same run (tracking the path and then stopping in front of the block). You will have three trials for this evaluation and the best one will be used to calculate your final project score. You will have 10 mins to finish these three runs (you can tune some parameters between each run, but you have 10 mins in total). You will need to generate three separate data recording files for each run based on the specifications defined in the project description. Make sure to name the file using the format – ECEXXX-group#_task1_run_#.txt – e.g. ECE456-group1_task1_run_1.txt. (or .rtf file if you are using LEGO Mindstorms).

Task 3 will be evaluated separately using a straight path. You will have three trials for this evaluation and the best one will be used to calculate your final project score. You will have **five mins** to finish these three runs (you can tune some parameters between each run, but you have five mins in total). You will need to generate three separate data recording files for each run based on the specifications defined in the project description. (ECEXXX-group#_task3_run_#.txt).

Evaluation Criteria

There are seven evaluation criteria:

1. Time: $C_1 = t_B - t_A$, the time your UV takes between once it starts moving from point *A* (t_A) to until it reaches point *B* (t_B) and fully stop. If $C_1 < 0.95 * \frac{path\ distance}{maximum\ speed}$, then you will go down two sub-grades.
2. Accumulated square tracking error: $C_2 = (\sum_{k=1}^{N} e_a^2(t_k))\Delta T$, where *N* is the total number of samples and ΔT is the sampling time in seconds. If $C_2 > \sum_{k=1}^{N} 441 * \frac{path\ distance}{maximum\ speed} \Delta T$, then you will go down two sub-grades.

   $e_a(t_k) = I_a(t_k) - I_{a,ref}$ is the tracking error for the anchor color/light sensor at each sampling time. $I_a(t_k)$ is the instantaneous intensity value of the anchor color/light sensor at each sampling time and $I_{a,ref}$ is the reference intensity value of the anchor color/light sensor.

   The reference intensity values are measured based on the initial value. **Make sure your UV is initially aligned to the track in such a way that your control error is zero.**

   In order to guarantee the fairness of project evaluation for path tracking task, you will need to write the sensor values to a local file in you EV3 in real-time with **a sampling rate smaller than 50ms** when you are doing the path tracking task. That is, the program should start recording the data at the same time as your UV starts moving and stop recording once your UV is fully stopped. The TA will then normalize all your sensor readings in to a range from 0-100 to calculate the C_2 value. **You will need to generate a different file for each trial with timestamp starting from start to end, then submit the data to TA after demonstration. Make sure to name the file using the format – ECE###-group#_task1_run_#.txt.** The data format should be:

   TimeStamp(ms),RLI,Ultrasonic

   ```
   '%d,%d,%d\n'
   ```

   Figure 5 gives an example of the data.

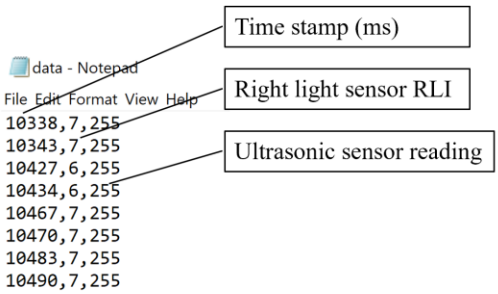


Figure 5. Data recoding example.

If your UV hits the block at the end of the path and/or keeps moving forward without stopping, we will use your C_1 value to extract the recorded data. For example, your C_1 is 15 second, we will extract 15/0.05 = 300 data points from the first recorded data set for your C_2 evaluation.

Note: If your average sampling rate is larger than 50ms, you will not get credit of the path tracking task.

Note: ECE 556 distant education students: You need to include the steps that are required to save and upload the file in the user manual. If your program needs to be modified during each run, please also state clearly in the manual. TA will only be able to modify the programs created using Matlab/Simulink and LEGO Mindstorms.

3. Tracking Smoothness: $C_3 = (\sum_{k=1}^{N} smooth_a(t_k))$.

   $smooth_a(t_k) = \left|\frac{I_a(t_k) - I_a(t_{k-1})}{\Delta T}\right|$ where, $I_a(t_k)$ is the instantaneous intensity value of the light/color sensor at sampling time t_k and $I_a(t_{k-1})$ is the instantaneous intensity value of the light/color sensor at sampling time t_{k-1} and ΔT is the sampling time of the system in seconds.

4. Distance to the block: $C_4 = \sqrt{(x_{UV}(t_B) - x_B)^2 + (y_{UV}(t_B) - y_B)^2}$ where $(x_{UV}(t_B), y_{UV}(t_B))$ is the position of your UV when it stops and (x_B, y_B) is the position of the block. Your UV cannot touch the block! Try to stop as close to the block as possible. If you touch the block at point B, you will automatically get zero points for C_4 in that trial.

5. Platooning mode error:

   To evaluate the platooning error, your EV3 program should start recording the data at the same time as you start the program and stop recording once your UV is fully stopped at the red tape.

   At the beginning, your UV should be placed 20 cm behind the UV_{TA}. Then you can start your EV3 program. Your UV should start moving as soon as the UV_{TA} starts moving at varying speeds.

   You will need to generate a different file for each trial with the timestamp starting from 0 and submit the data to TA after demonstration. The data format of the light sensor reading should be the same as the task 1.

   Ideally, if your UV can perfectly follow the UV_{TA}, the distance $d(t_k)$ between your UV and UV_{TA} will always be a constant, so C_5 is calculated as the standard deviation of your data set:

$$C_5 = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(d(t_k) - \mu)^2}$$

where $d(t_k)$ is the distance between you UV and UV_{TA} at each sampling time t_k and μ is the reference distance between you UV and UV_{TA}, the initial reading of your ultrasonic sensor is used as the reference, since your UV is positioned 20 cm behind the UV_{TA}. N is the number of data points. C_5 indicates whether your UV is maintaining a constant distance from the front UV_{TA}.

6. Red tape stop mark: C6 = 100%, if the UV stops at the destination red mark; C6= 0, if the UV fails to stop at the red mark.

7. Peer-review: C7 is used to incorporate the peer-review among the group members. Every week each group member will need to fill a form to evaluate his/her other group members. We will not share your form with others.

The following weights will be used for C_1, C_2, …, C_7 to calculate your final project grade:

C_1: 15%, C_2: 20%, C_3: 20%, C_4: 15%, C_5: 20%, C_6: 5%, C_7: 5%.

Example Evaluation Code

Based on the evaluation criteria above, the TA has written a MATLAB to help you evaluate the cost functions while you are tuning your controller. Please use the 'EV3_eval.p' code provided on the Moodle website to evaluate your performance: (make sure your source data is under the same folder as the function is)

```
[C1, C2, C3, C5] = EV3_eval('data.txt');
```

This is a protected MATLAB function. The source code is published in the Appendix A.

Some typical performance example on 4 typical tracks are given in the T06 slides for your reference.

Algorithm to Incorporate your Project Grade into the Course Grade

1. The TAs will calculate the average of Exam 1 and Exam 2 [i.e., (Exam1 score + Exam 2 score)/2] and then find the maximum and minimum of the Exam averages among all students of your session as the reference points for the normalized scores (S_max = Exam_max, S_min = Exam_min).
2. The TAs will find the maximum and minimum (C_max and C_min) for each of the project measures (C_1, C_2, C_3, and C_4) among all groups in your session, and find the linear relation with C_max and C_min.
3. The TAs will then calculate your normalized scores (S_1, S_2, S_3, and S_4) for each project measure by linearly mapping the interval [C_max, C_min] to the interval [S_min, S_max]. An example of calculating the normalized score S_1 for measurement C_1 is shown below:

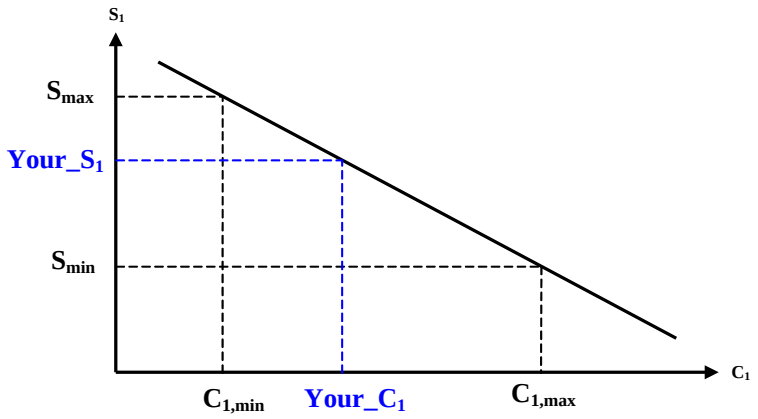


Figure 6. Linear mapping from [C_max, C_min] to the interval [S_min, S_max] for C1.

Thus,

$$S_i = \frac{1}{C_{i,min} - C_{i,max}}((S_{max} - S_{min})C_i + S_{min}C_{i,min} - S_{max}C_{i,max})), i = 1{:}\,5,7$$

Note: $S_6 = C_6$, that is, if your C_6 = 100%, your S_6 = 100.

4. The rest is to incorporate your normalized scores for all 7 measures into the final project grade.

$$G = 15\%S_1 + 20\%S_2 + 20\%S_3 + 15\%S_4 + 20\%S_5 + 5\%S_6 + 5\%S_7$$

Project Report

Each group will need to turn in a formal report, which is less than 20 pages, with the following topics BEFORE project demonstration:

1. Introduction (~ 1 page)
2. Project Setup (~ 3 pages)
3. Algorithm design (~ 3 pages)
4. Algorithm development (~ 2 pages)
5. Results (~ 3 pages)
6. Discussions (~ 3 pages)
7. References

Note: The project report is not included in the project grading. However, if your project demonstration goes wrong, the TAs will grade your project based on your project reports.

Appendix A

```
function [C1, C2, C3, C5] = EV3_eval_ver2(filename)
% Outputs
% C1 : lap time (ms)
% C2 : tracking error
% C3 : tracking smoothness
% C5 : Platooning performance

% initialize costs
C1 = Inf; C2 = Inf; C3 = Inf; C5 = Inf;

%% Main Code
try
    % Read-in raw data
    [fPath, fName, fExt] = fileparts(filename);
    switch lower(fExt)
      case '.txt'
        data = readtable(filename);
      case '.rtf'
        fID = fopen(filename, 'r'); % Open our data file
        Temp = zeros(1,3);
        index = 1;
        while ~feof(fID) % Loop until we reach the end of the file
            tline = fgetl(fID); % Read in a line of data, discard it
            Temp(index,:) = sscanf(tline,'%f,%f,%f');
            index = index + 1;
        end
        fclose(fID);
        data = table(Temp(:,1), Temp(:,2), Temp(:,3));
      otherwise  % Under all circumstances SWITCH gets an OTHERWISE!
        disp('Unexpected file extension: %s', fExt);
    end

    time = data.(1);
    RLI = data.(2);
    Ultra = data.(3);
    % Time table
    if time(end) > 180
        ttEV3_raw = timetable(milliseconds(time),RLI,Ultra,...
        'VariableNames',{'RLI','Ultra'});
    else
        ttEV3_raw = timetable(seconds(time),RLI,Ultra,...
        'VariableNames',{'RLI','Ultra'});
    end

    % provide summary of the data points
    disp('============Raw Data Statistics==================');
    summary(ttEV3_raw)

    % Check for data validaty
    if median(diff(ttEV3_raw.Time))> milliseconds(50)
        disp('Sampling time greater than 50 ms! -> Disqualify')
        return
    end

    % normalize to 50 ms sampling time for grading
    ttEV3_regular =
retime(ttEV3_raw,'regular','linear','TimeStep',milliseconds(50));
    % provide summary of the data points
```

```
    disp('============Normalized Data Statistics============');
    summary(ttEV3_regular)

    %% C1
    C1 = milliseconds(ttEV3_raw.Time(end) - ttEV3_raw.Time(1));

    %% C2
    % Normalize the sensor readings to (0 ~ 100) range
    normlized_Anchor = ttEV3_regular.RLI.*(100/max(ttEV3_regular.RLI));
    % Use initial sensor reading as the reference value
    Anchor_ref = normlized_Anchor(1);
    % Tracking error
    error = normlized_Anchor - Anchor_ref;
    % Data sampling time
    delta_T = 0.05;
    % Calculate C2
    C2 = sum((error.^2).*delta_T);

    %% C3
    % calculate C3
    C3 = sum(abs(diff(normlized_Anchor)./delta_T));

    %% C5
    % Use initial sensor reading as the reference value
    ultra_ref = Ultra(1);
    ultra_error = ttEV3_regular.Ultra.*(20/ultra_ref) - 20;

    C5 = sqrt(mean(sum((ultra_error.^2))));

    disp('=========================================================');
    fprintf('Lap time (ms) C1 is: %d\n',C1);
    fprintf('Accumulative square tracking error C2 is: %.2f\n',C2);
    fprintf('Tracking smoothness C3 is: %.2f\n',C3);
    fprintf('Platooning RMSE C5 is: %.2f\n',C5);

catch
    disp('Bad input data. Check your input data.');
end
end
```