`

Schulich School of Engineering
University of Calgary

ENGG 233 –Fall 2018

Department of Electrical and Computer Engineering

Term Project: Digital Dashboard

Project Objectives

1. To gain experience with the design and development of a relatively larger software project
2. To gain practical experience in using programming constructs learned in ENGG 233 such as classes, arrays, functions, etc.
3. To gain experience with iterative development process
4. To develop teamwork and presentation skills as a programmer

Submissions and Group Work

Students have the option of working on the project in groups of two or working alone. Groups of larger than 2 people will not be allowed.

If students choose to work in groups, **they must choose a partner from within their TA lab groups.** For example, if a student is in B03-Group Blue, they must choose a partner from that group.

Instructions for Groups

Students who work in groups must follow the following guidelines and regulations:

1. The workload **must be divided equally** between the two students. During the demos, the TAs will specifically ask about what each student has done in order to ensure both students have contributed.
2. Both students must be present at each demo. If a student is absent, that student will not be awarded the grades for the demo.
3. Only one of you needs to submit each milestone to D2L. However, please make sure both of your names are included in your submission.

Important Note:

You can work with other groups and students, however, be very careful not to simply use the code from other students. Plagiarism is a serious academic offense. Please consult the University of Calgary Calendar for more information:
http://www.ucalgary.ca/pubs/calendar/current/k-5-1.html

`

Introduction

You have been hired by a smart automobile company to write and design their new digital dashboard for consumer cars. This dashboard replaces the legacy analog dashboard with an intuitive and eye-catching design that is easy to modify and add features to with future software updates. The dashboard displays various information like current speed and fuel level based on sensory inputs from different modules of the car.

You are responsible to write the main processing software and the user interface of the digital dashboard. The digital dashboard should be able to display useful information to the driver based on the input sensory data provided. There are four inputs available: current Revolutions Per Minute (RPM) of the engine, current fuel level of the fuel tank, the current gear ratio which is based on the gear number and the position of the car (x and y). These inputs are provided periodically every second. At each time step, the values for these inputs are read by the digital dashboard program and should be used to calculate the following metrics:

- **Speed** of the car: The radius of the wheel is assumed to be known (in meters). Based on the two inputs: current gear ratio and revolutions per minute of the engine, speed can be calculated for each time step using the following equation:

  $$speed(meterpersecond) = \frac{RPM}{60} * \frac{1}{GearRatio} * 2 * \pi * Radius. \quad (1)$$

  In the above equation, $2 * \pi * Radius$ is the perimeter of the wheel. Gear ratio represents the relationship of the number of revolutions per minute of the engine to revolutions per minute of the wheels. Multiplying revolutions of the wheel by its perimeter calculates the distance travelled in the period.

- **Travelled Distance**: Having calculated the speed in meters per second, the distance travelled during a second in meters is equal to current calculated speed.
- **Fuel Economy**: Fuel economy is the distance travelled per unit volume of fuel used. Each time step, the current level of the fuel tank is available. Current fuel economy is calculated by dividing the distance travelled in a second by the difference between the current fuel level and previous fuel level in liters (fuel consumed during the past second). The equation to calculate the fuel economy is:

  $$FuelEconomy(km/liter) = \frac{DistanceTravelled(km)}{ConsumedFuel(liter)} \quad (2)$$

- **Range**: Range is the estimated number of kilometers the vehicle can go before running out of fuel. To calculate range, use the average fuel economy during the past 60 seconds (if available) as follows:

  $$Range(km) = AverageFuelEconomy(km/liter) * RemainingFuel(liters)(3)$$

  Note: If all the last 60 second of data is not available, use the data that is available so far.

- **Fuel Consumption:** Fuel consumption is the average fuel consumed by car to travel 100 km and it is calculated as follow:

$$\textit{Fuel Consumption}(\textit{liter}) = \textit{1/Average Fuel Economy (km/liter) * 100 (km)} \qquad (4)$$

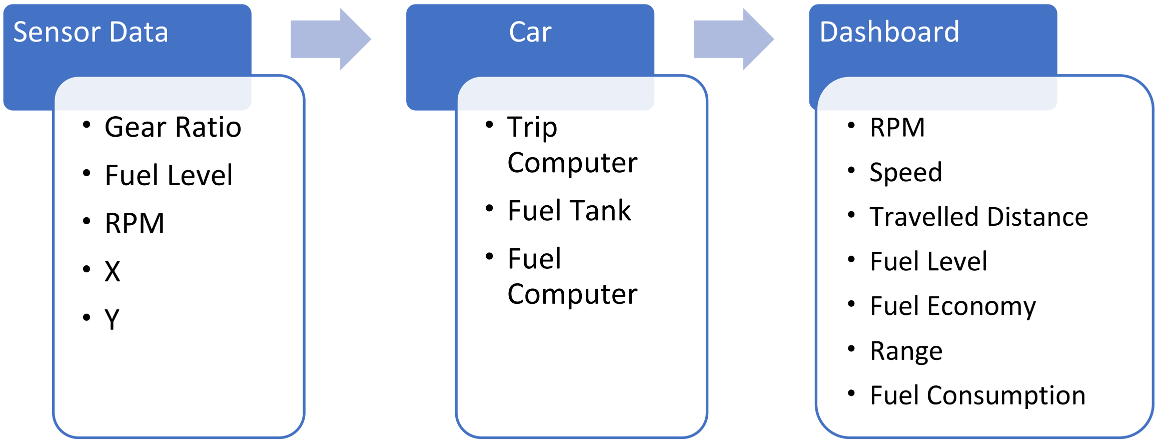


Figure 1 Main components of the solution

The above figure shows the main components of the solution. Each second, the input data is read from a file. This is simulating real world scenario where inputs are constantly provided to the processor. The data is processed by three main components of the car, namely Trip Computer, Fuel Tank and Fuel Computer. After performing the calculations for a second, the parameters like RPM, speed, etc. are displayed on the digital dashboard. The final solution is a screen that contains several gauges to display car status. The instructions on how to structure and implement the digital dashboard code is provided in the rest of this document.

Minimum Requirements

- Interface requirements:

- A simple menu allowing the user to choose between different car data.
- A dashboard (shown in figure 1) consisting of:
    - **Speedometer**: A gauge for displaying the current speed in km/h
    - **Tachometer**: A gauge for displaying the current engine RPM
    - **Fuel gauge**: A gauge for showing the current fuel level of the tank in liter
    - **Fuel Economy**: A text field showing the fuel economy in km/liter
    - **Odometer**: A text field for displaying the total distance travelled by the car in km
    - **Range**: A text field for showing the distance (km) car can travel on the remaining fuel in the fuel tank.

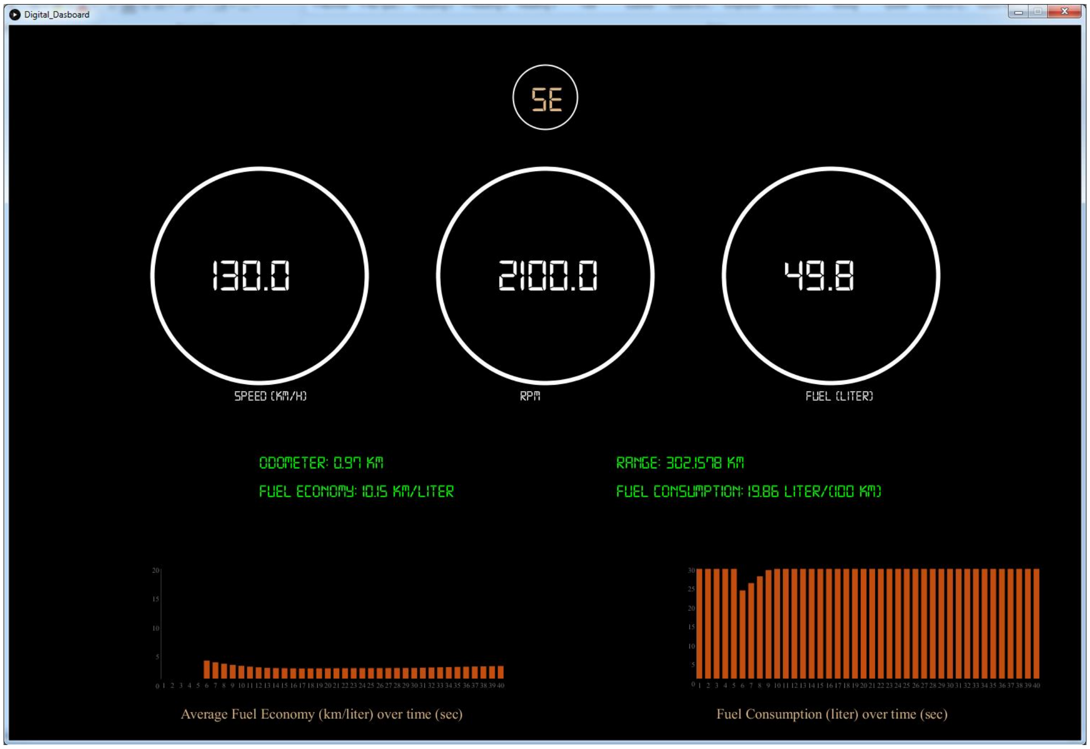
`

- **Fuel Consumption**: A text field for showing the average fuel consumption for travelling 100 km.

- **Direction:** A simple indicator specifying the direction of the vehicle based on the input location data.

- **Average fuel economy history:** A chart displaying the history of the last 40 calculated average fuel economy values. The average fuel economy (km/liter) includes the most recent 60 seconds (if available).

- **Fuel consumption history:** A chart displaying the last 40 calculated fuel consumption values.



SE

130.0

SPEED (KM/H)

2100.0

RPM

49.8

FUEL (LITER)

ODOMETER: 0.97 KM

FUEL ECONOMY: 10.15 KM/LITER

RANGE: 302.1578 KM

FUEL CONSUMPTION: 19.86 LITER/(100 KM)

Figure 2 An example of a completed digital dashboard

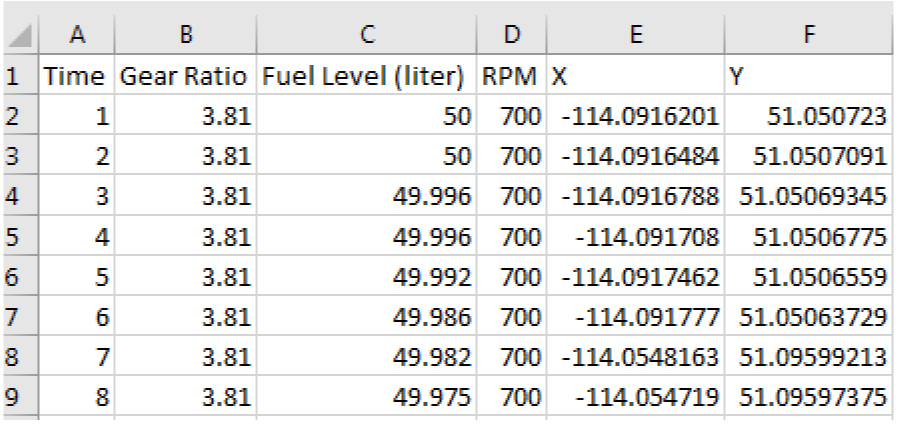
- Programming Module Requirements

- A Gauge module that can be re-used to display different numbers on the dashboard.
  - Gauge needs to display current value, unit of the parameter as well as the minimum and maximum of the possible value.
  - Gauge should display the values in the first and last 15% of the range differently. For example, you can use different colors for this purpose.
- A TripComputer that calculates speed and distance travelled based on RPM and Gear Ratio. Details of calculations are provided in equation (1).

- A FuelTank that calculates the remaining fuel using the input fuel level.
- A FuelComputer module that calculates fuel economy and range based on the outputs of trip computer and fuel tank: distance travelled, fuel consumed and fuel remaining.
- A SensorDataProvider module that reads the input sensor data from a comma separated format (csv) file provided. Two following two files will be provided that contain the sensor information. The application should prompt the users to select the desired file when the program starts. Each file contains sensor data from the time that the car starts to work. Use the provided numbers in the table for initializing the car properties depending on the selected file.

Vehicle	Data File Name	Wheel Radius (cm)	Tank Capacity (liters)
minicar	car_status_BMW_323i.csv	23	60
truck	car_status_Truck_F150.csv	25.4	80

Time (second), Fuel Level (liters), RPM, Gear ratio, X (latitude) and Y (longitude) for every second is stored in the file. The files will be provided for you and your dashboard application should load it at the beginning depending on user selection and then in each iteration should read one line and provide the input data to main classes. Once the stats like speed and fuel economy are calculated, the results are displayed, and the application moves to the next line of data. This is a screenshot of the file opened in Excel:



	A	B	C	D	E	F
1	Time	Gear Ratio	Fuel Level (liter)	RPM	X	Y
2	1	3.81	50	700	-114.0916201	51.050723
3	2	3.81	50	700	-114.0916484	51.0507091
4	3	3.81	49.996	700	-114.0916788	51.05069345
5	4	3.81	49.996	700	-114.091708	51.0506775
6	5	3.81	49.992	700	-114.0917462	51.0506559
7	6	3.81	49.986	700	-114.091777	51.05063729
8	7	3.81	49.982	700	-114.0548163	51.09599213
9	8	3.81	49.975	700	-114.054719	51.09597375

Figure 3 Sample input data

- The main program that you can call it DigitalDashboard.

Important note: When one simulation ends, the user should be promoted to start another simulation (i.e. using the menu) or quit the program.

Optional Requirements for Bonus Points

Students may add additional features/functionalities for a maximum of 15 bonus marks. These would include but not limited to:

`

- Creative visualization
- Visualization of the history of fuel economy as a graph
- Creative visualization of the direction of the vehicle
- etc.

Program Modules and Classes

Your program must have several classes that will be organized as separate tabs in Processing IDE. Here is the list of some of the modules or classes that your program should have (Note: you may decide to have more classes. You are also free to make changes to the classes as you see fit, as long as the functionality of the program is not compromised):

- Class **SensorDataProvider**

  This class has the following member variables:

  - filePath to store the path of the selected sensor data file
  - dataTable of type *Table* to load the data in the csv input file
  - currentIndex representing the data row index that needs to be processed

  This class has the following member functions:

  - Initialize(): This function is called at the beginning of the program. It reads the whole input csv file and loads it into dataTable. For more information about reading a csv file in processing please refer to https://processing.org/reference/loadTable_.html.
  - readNext(). This function is called in the main loop of the program (draw). Each call advances to the next line by incrementing currentIndex. In order to make sure that the program does not crashes because of null reference, there should be a cap on incrementing currentIndex.
  - readRPM(), readFuelLevel(), readRatio(), readX() and readY() functions to read the current row of the file.

- Class **Gauge**

  This class has member variables as follows:

  - Member variables related to the displaying parameter: Minimum, Maximum, Current Value and Units.
  - Displaying member variables: position, size, etc.

  Required member functions:

  - A function that receives the current value as parameter (argument) and sets it. It is required to check the value of the parameter for validity (between minimum and maximum) before setting the current value.
  - A function named display that draws the gauge on the screen.

- Class **TripComputer**

  This class is responsible for calculating the speed and distance travelled based on RPM and Gear Ratio.

  Member variables:

  - RPM
  - gearRatio
  - totalTravelledDistance
  - etc.

  Member functions:

  - getCurrentSpeed
  - updateTotalDistance
  - etc.

- Class **FuelTank**

  This class represents the fuel tank. Based on the input current tank level (read from file) and the capacity of the tank, it calculates the remaining and consumed fuel.

  Member variables:

  - Tank Capacity
  - Fuel Level
  - etc.

  Member functions:

  - getConsumedFuel: Consumed fuel is calculated by subtracting current level from previous level.

`

  - etc.

- Class **FuelComputer**

  This class calculates the car fuel economy based on two input parameters, the calculated distance travelled, and fuel consumed. It also calculates the range using equation (3). The recent fuel economy values and its average must be stored in memory and then used for calculating the average fuel economy.

  Member variables:

  - fuelEconomy
  - fuelConsumption
  - fuelEconomyHistory
  - range
  - etc.

  Member functions:

  - calculateFuelEconomy
  - calculateAverageFuelEconomy
  - calculateFuelConsumption
  - etc.

- Class **Car**

  This class holds an instance of three computing classes as member variables:

  - A TripComputer object
  - A FuelTank object
  - A FuelComputer object

  It has one member function called processInput that accepts sensor data and passes the data to computing classes by calling their member functions like setRPM, setGearRatio, etc.

`

Project Requirements in Iteration

Software projects can be very complex, therefore the process used to manage this complexity is very important. One effective process widely used in software development is referred to as iterative development.

Simply put, iterative development is the processes of breaking down a larger project into smaller steps (i.e. iterations or milestones). To ensure you are on track for the project, and to make sure you will get marks for all your efforts, not just for the completed project, this project is divided into 2 different milestones. For each milestone, you should fully test your code and ensure you satisfy all the requirements for that milestone correctly. This is very important as you don't want to keep building on a faulty project.

Important Advice: You should save a version of your project after each milestone. This is the basic idea of "*version control*" in real-life software development. This is to keep track of the changes made to your program and to ensure that you have a working version of your software which you can go back to if needed.

Milestone 1 (45 points):

This milestone is due on the week of Nov 26th. Students must demo their milestone in their scheduled lab during this week, and submit their code on D2L before their scheduled labs:

- **Lab Section B01:** Friday , Nov 30, before 9:30 AM.
- **Lab Section B02:** Wednesday, Nov 28, before 3:00 PM.
- **Lab Section B03:** Friday, November 30, before 2:00 PM.
- **Lab Section B04:** Thursday, November 29, before 12:00 PM.

In this milestone you are expected to implement a simple version of the program which include calculations for Speed

- Speed
- RPM
- Fuel level

At this state, the dashboard would look fairly simple. A program output, as demonstrated in the following figure, is the minimum expectation; but you can demonstrate additional feature if you like:

`

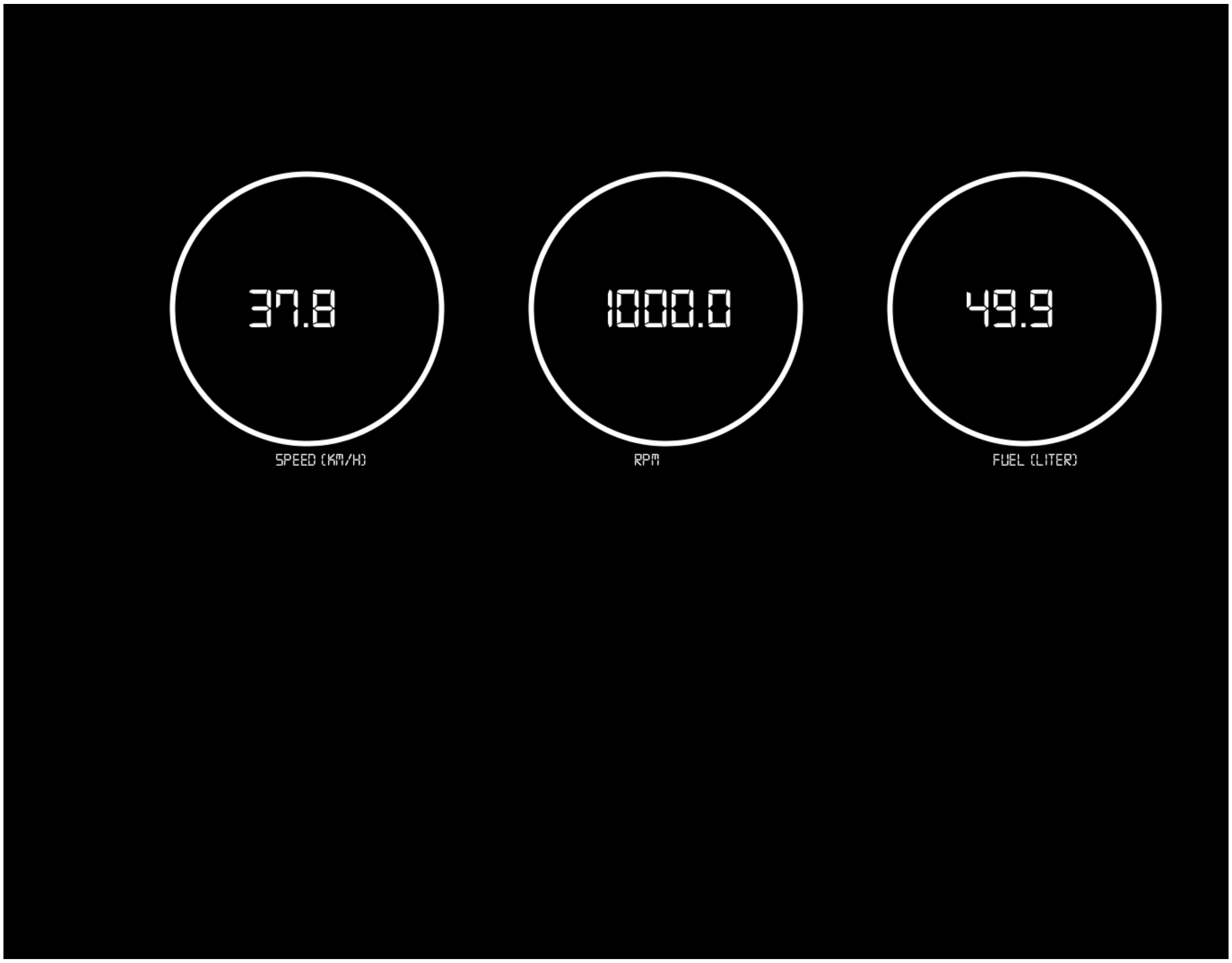


Figure 4 - Simple version of the dashboard

What to Submit for Milestone 1:

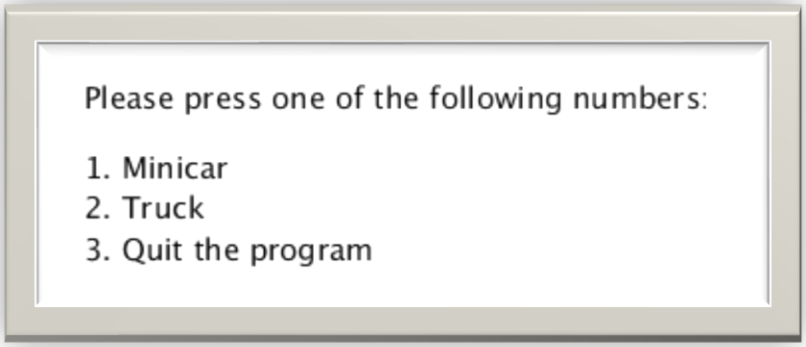
Submit a .zip file containing all .pde files you have created in this milestone to the dropbox on D2L.

Milestone 2 (55 points):

This milestone is due on the week of December 4th. Students must demo their milestone in their scheduled lab during this week, and submit their code on D2L before their scheduled labs:

- **Lab Section B01:** Friday , December 7, before 9:30 AM.
- **Lab Section B02:** Wednesday, December 5, before 3:00 PM.
- **Lab Section B03:** Friday, December 7, before 2:00 PM.
- **Lab Section B04:** Thursday, December 6, before 12:00 PM.

`



Please press one of the following numbers:

1. Minicar
2. Truck
3. Quit the program

Figure 2: Simple version of the Menu

In this milestone you are expected to implement the rest of the project including:

- Menu for choosing the car (Figure 5 above)
- Odometer (total distance traveled)
- Range
- Fuel Consumption
- Data visualization of the two charts in figure 1
- Direction

The output of your program (i.e. your dashboard) would look similar to Figure 1 above.

What to Submit for Milestone 2:

Submit a .zip file containing all .pde files you have created for this project to the dropbox on D2L.