COMPRO1 (for S18A and S19A) MP Specifications – Part 3 of 3
Application of Cosine and Sine User-Defined Functions
Part 3 is 40% of the MP Grade

I. INTRODUCTION

The 3rd part of the machine problem is an **application of the cosine and sine functions** that you yourself defined. The application in this case is concerned with a robot movement simulation.

"Robby" is a simple autonomous mobile robot that is able to move in a two-dimensional (2D) space. The top view of the robot is shown in Figure 1. The dot represents the robot's position, and the arrow represents the forward direction of the robot when it moves. The opposite direction of the arrow represents the backward direction.

Without loss of generality, let the 2D space be the 2D Cartesian X-Y plane where the X axis will be drawn along the horizontal and the Y axis will be drawn along the vertical. The X and Y axes are perpendicular to each other, and they intersect at a point that we know as the origin. The origin coordinates are (0, 0) where the first zero is the X coordinate, and the second zero is the Y coordinate.

Robby's **position** and **orientation** are to be tracked at all times. For simplicity purposes, the robot's position is computed as a point on the robot's body – more specifically its centroid. Let the robot position be represented by the coordinates `(fRobotX, fRobotY)` measured with respect to the origin. The black dot in Figure 1 represents Robby's position.

Robby's orientation, on the other hand, is measured as the angle that the forward direction arrow makes with a line that is parallel to the X axis and passing through its centroid. In Figure 1, the angle is 0 degree. Figure 2 shows the robot with an orientation of 45 degrees, while Figure 3 shows the robot with an orientation of –60 degrees (negative).

Note that the angle is positive if it is measured counterclockwise from the axis parallel to the X axis (shown as dashed line in Figures 2 and 3) towards the forward direction arrow. Otherwise, it is negative if it is measured clockwise from the dashed line towards the forward direction arrow, as in the case of Figure 3. Let the robot orientation be represented by the angle `dRobotAngle`. The range of values that can be assumed by `dRobotAngle` is –360.0 to 360.0 degrees.

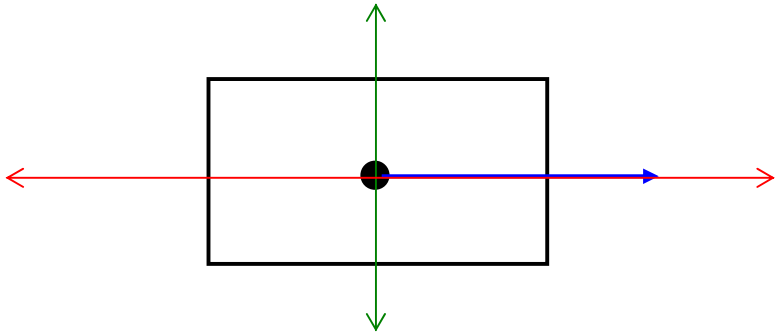

Figure 1: Top view of Robby in its initial position (0,0) and orientation (0 degree). The dot represents the robot's centroid position denoted as (fRobotX, fRobotY), while the blue arrow represents its forward direction. The X and Y axes are drawn in red and green colors respectively. The orientation dRobotAngle is 0 degree.

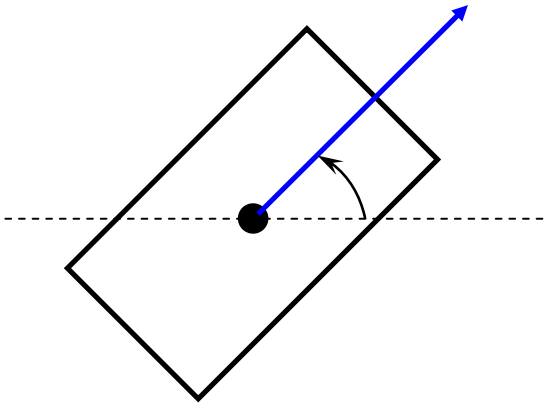

Figure 2: Robby with an orientation of dRobotAngle = 45 degrees.

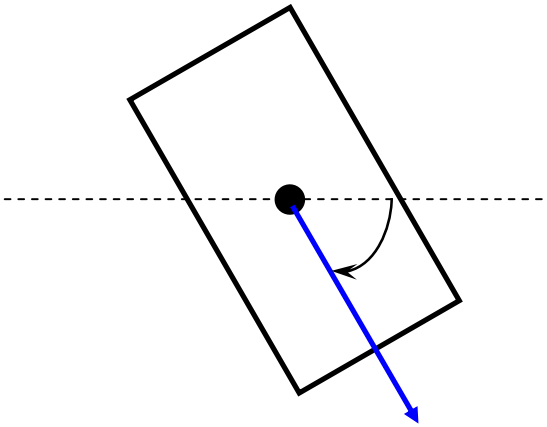

Figure 3: Robby with an orientation of dRobotAngle = -60 degrees.

Note that initially, Robby is positioned at the origin, i.e., **`fRobotX`** is 0, and **`fRobotY`** is 0. The initial orientation of the robot, **`dRobotAngle`** is 0, i.e., its forward direction is in the same direction as that of the positive X axis. **The robot's position and orientation are referred also as the robot's state variables.**

The robot can be moved in two ways, namely:

1. Translation – i.e., the robot can move forward or backward, and
2. Rotation – i.e., the robot can rotate counterclockwise or clockwise.

Note that a translation does not change the robot's orientation. Also, a rotation does not change the robot's position.

The following shows an example of how the movements affect the robot's position and orientation. Assume that the robot is currently in its initial position (0, 0) and orientation of 0 degree.

1. If we make the robot move forward by a distance of 10 units, the robot's position will become (10, 0), i.e., **`fRobotX`** is 10, and **`fRobotY`** is 0.
2. Next, we can make the robot rotate counterclockwise by 90 degrees. So, from its current orientation of 0 degree, the robot's new orientation will become 90 degrees (i.e., **`dRobotAngle`** will become 90). Note, that this orientation corresponds to a robot whose forward direction now is

in the same direction as positive Y axis.

3. Next, if we issue a forward movement with a distance of 25 units, the robot will be moved from (10, 0) to its new position. Computations will yield a new position of (10, 25).
4. Next, rotate the robot clockwise by 30 degrees. From its current orientation (which is 90 degrees), the robot's new orientation will become 60 degrees (i.e., **`dRobotAngle`** will become 60).
5. Next, if we issue a forward movement with a distance of 12 units, the robot will be moved from (10, 25) to its new position. Computations will yield a new position of (16, 35.39).
6. Next, rotate the robot clockwise by 20 degrees. From its current orientation (which is 60 degrees), the robot's new orientation will become 40 degrees (i.e., **`dRobotAngle`** will become 40).
7. Next, if we issue a backward movement with a distance of 6.5 units, the robot will be moved from (16, 35.39) to its new position. Computations will yield a new position of (11.02, 31.21).

It is important that you know and understand SIMPLE trigonometry in order to understand the effects of translations and rotations. It is strongly suggested that you do the manual computations (using a calculator) first to simulate and confirm the example above before proceeding to the next page.

II. REQUIREMENTS

Your task is to define some functions for a **simulation program** that will control the movement of the robot. The simulation program is command based, i.e., the user has to supply some external input to control the robot. The command codes (integer data type) and the corresponding semantics are given in the following table:

Command Code	Semantic
1	**Initialize/Reset Robot Status** Call the function `InitializeReset()` which will set/reset the robot's position to (0, 0) and the robot's orientation to 0 degree.
0	**Display Status Command Code** Call the function `DisplayStatus()` which will output the robot current state, i.e., its current position and orientation (in degrees).
8	**Forward Translation Command Code** If the user issued this command, the program should: 1. Ask the user to input a single precision floating point value representing how much distance (relative to its current position and orientation) Robby will move forward 2. Call the function `TranslateForward()` to update the robot's new position
2	**Backward Translation Command Code** If the user issued this command, the program should: 1. Ask the user to input a single precision floating point value representing how much distance (relative to its current position and orientation) Robby will move backward 2. Call the function `TranslateBackward()` to update the robot's new position
9	**Counterclockwise Rotation Command Code** If the user issued this command, the program should: 1. Ask the user to input a double precision floating point value representing the angle of rotation in **degrees** 2. Call the function `RotateCounterClockwise()` to update the robot's new orientation For example, if the current orientation is 45 degrees, and the user rotates the robot counterclockwise by 15 degrees, the new orientation will become 60 degrees.
3	**Clockwise Rotation Command Code** If the user issued this command, the program should: 1. Ask the user to input a double precision floating point value representing the angle of rotation in **degrees** 2. Call the function `RotateClockwise()` to update the robot's new orientation For example, if the current orientation is 45 degrees, and the user rotates the robot clockwise by 15 degrees, the new orientation will become 30 degrees.
4	**Quit Simulation Command Code** Call the function `Quit()`, then terminate program execution.

You will be provided three files for this activity, namely:

a. **`mp3_robot.h`** – contains related **`#defines`** and function prototypes
b. **`lastname_mp3_robot.c`** – contains the skeletons of the functions in the header file above
c. **`lastname_mp3_main.c`** – contains the skeleton of the **`main()`** function

Your task is to implement the body of the function definitions in the C skeleton programs (items b and c in the list above). **Make sure that you read and follow the information specifically written inside the above-mentioned files.**

You will need to compile the C programs separately and link them together with the object file containing the **`cosine()`** and **`sine()`** functions from Part 2 of the MP in order to produce the executable file. Read the contents of **`lastname_mp3_main.c`** for details.

III. DELIVERABLES AND SUBMISSION DEADLINES

You need to submit THREE items, namely:

1. C program source codes for three files: **`lastname_mp3_robot.c`**, **`lastname_mp3_main.c`** and **`lastname_mp2_math.c`**. Make sure that you followed the naming convention. The softcopy of the source codes should be received as email attachment files **before 11:59PM of August 18, 2014 (Monday)**. Send your email to the **TWO** addresses indicated below:

   `bong.salvador@delasalle.ph`
   `pulingfe@yahoo.com`

   Email subject should be:

   COMPRO1 LASTNAME<underscore>FIRSTNAME <space> SECTION <space>MP3

2. Accomplished MP submission checklist (print and accomplish the document named "mp3_checklist.pdf").

3. Hardcopy/printout of your C program source codes.

Staple the DOW on top the hardcopy of your program. Submit these **on August 19, 2014 (Tuesday)** within the **first 10 minutes of your own COMPRO1 class hours only** (not those of another section). Note that the softcopy of the source code must be exactly the same as the hardcopy. That is, DO NOT modify your program after submitting it via email.

INCOMPLETE SUBMISSION (i.e., did not email the source codes, or did not submit the checklist, or did not submit the hardcopy of source code or sample output) will automatically result into a grade of 0.0 for this particular requirement.

LATE SUBMISSION WILL **NOT** BE ACCEPTED unless there is a valid and verifiable excuse (example: sickness, emergency).

IV. TESTING, CHECKING and GRADING SCHEME

Your **`lastname_mp3_robot.c`** and **`lastname_mp2_math.c`** files will be compiled separately to produce the object files. Thereafter, they will be linked with the object file of a "nice" program that I created for testing purposes. Your programs should compile, link and run properly (i.e., 100% correct) to earn a perfect score for this requirement.

***** End of the Machine Problem (Part 3) Specifications *****

Last Note: Consult me immediately if you have any clarification, question or problem regarding this academic activity. Enjoy! ☺☺☺