EECE 6354: Advanced Real-Time Systems

Class Project

Part I Instructions

Tournament Date: October 16, 2017

1 Project Description

For this part of the project, you are assigned the task of developing an automated missile-avoidance system. You will have control over one tank which will be under fire from at most three missiles at any given time. The tank is equipped with a GPS unit, allowing you to know your global position, as well as radar, enabling you to locate other objects (missiles in this part of the project) within the sensing range. Your radar can inform you of the number of missiles within its sensing range, as well as the distance and direction of each missile with respect to the *center* of your tank and its current orientation.

The objective is to ensure that your tank is hit by as few missiles as possible. The student with the lowest number of tank hits after five minutes will be declared the winner. Note, however, that for every minute your tank is in the field, its sensors degrade, resulting in a smaller and smaller sensing area. Thus, avoiding missiles will get progressively harder as time goes by.

2 Distributables

- ***ClassProject.eww* and associated directories**: These files comprise the IAR workspace and environment needed to implement your avoidance system. The only existing files you are allowed to alter are 'app.c' and 'app_cfg.h'. You may add additional files as you see fit.
- ***ClassProject/Students/FirstInitialAndLastName_app.c***: This file should be used as your own personal 'app.c' file. You will need to rename this file using the convention "Your first initial + Your full last name"_app.c with correct capitalization (Ex: my file would be named 'ALedeczi_app.c'). After you have done that, open the project workspace, right-click the 'ClassProject→APP→Students' group, select 'Add→Add Files', and select the file you just renamed. From now on, you will only add your code to this file.
- ***BattlefieldGUI.jar***: This Java program is used to provide a visual reference of what is happening during the execution of your program. It requires that you connect your computer to the evaluation board using a serial (RS-232) port or a USB-to-Serial converter. (This is necessary for your program to run, so try to locate such a cable as soon as possible.)

3 Implementing Your Code

When you open the IAR workspace, you will see a file named **BattlefieldLib.h** in the APP directory. This contains the user API to the project framework. It shows you all the available functions you can use to get information about your tank, control its movements, or poll its sensors. You can safely ignore anything related to a turret or firing for this part of the project. The first time you open the workspace, you will need to open 'app.c' and rename the 'extern void...' prototype and corresponding function call in **main()** to match your personal application file as described in the *Distributables* section (Ex: 'ALedeczi_init(void)'). Once completed, you may close this file and never use it again. Note that you will also need to rename the initialization function in your personal application file.

To begin your avoidance system, you will need to create one or more static task functions which will implement your desired functionality. Note that you will **NOT** create these tasks using **OSTaskCreate()**, but rather will pass them as pointers to the project framework using the **register_user()** function as outlined in *BattlefieldLib.h*. When you register each function, you are allowed to specify the priority at which you would like it to run, as well as pass a pointer to any parameters you would like to have access to in the task. Two important things to note:

1. Your tasks' priorities must be between **APP_TASK_USER_PRIO** and 8 for them to register correctly. You are encouraged to play with adjusting priorities to take advantage of the preemptive nature of the RTOS.
2. Any parameters passed to your task are passed as pointers; thus, they MUST remain in scope for the duration of the task's execution. If you experience application freezing or tank problems, ensure that all of your user-defined variables are correctly instantiated and remain in scope.

If you need to use any mutexes, semaphores, flags, or other operating system-specific constructs (and you should), make sure you initialize them first in the **YourName_init()** function. Additionally, if you need to use the built-in random number generator, **rand()**, it has already been seeded, so do NOT reseed it by calling **srand()**. Finally, as a general note, if it seems like you have one or more tasks that are never being run, make sure that each and every task you create either sleeps or pends at some point during its execution. This will ensure that the scheduler is called and your other tasks get the opportunity to run.

4 Project Framework

The project framework is set up to look like a large battlefield with bounds **MINX**, **MAXX**, **MINY**, and **MAXY**. It will not allow you to go out of those bounds, so to avoid getting stuck, you will want to be aware of your position relative to the walls and take pains to avoid them. Also, when creating your tank, you are not allowed to decide its starting position or orientation. This is to ensure that your algorithm is truly adaptive and robust, and to discourage hard-coding of specific avoidance maneuvers. It is also important to keep in mind that the field is implemented in pixel coordinate space, meaning that (0,0) lies in the upper left-hand corner of the battlefield. This is important when performing mathematical operations on global coordinates as many operations must be inverted to correctly calculate the desired answer.

Global orientation is represented in positive radians from 0 to 2π, with 0 indicating that your tank is facing directly to the right. As your tank's heading revolves counter-clockwise, its orientation value increases. Thus, whenever you call a function to find your current heading, it will return a positive **float** in this range. Likewise, whenever you specify a heading, you will need to restrict it to these bounds.

To indicate a change in steering, you will call the **set_steering** function with the relative desired wheel angle as a parameter. You must restrict this angle to between -STEER_LIMIT and STEER_LIMIT (or the framework will do it for you). A value of 0 indicates that your wheels are facing directly forward. Positive values indicate that they are oriented to the left, while negative values indicate that they are oriented to the right. This may feel counter-intuitive, but it is in keeping with the standard convention that positive angles increase in the clockwise direction.

5 Tips

- The project framework requires the Java GUI to initiate movement. Your task will simply pend execution until you have started the client, connected to the serial port, and clicked the 'Start' button. Thus, ensure that the code on your evaluation board is downloaded and running (not stuck in debug mode at **main()**) before starting the Java GUI.
- Your tank's radar can only return the relative positions of each of the missiles. It may be beneficial to know their global positions and/or headings to know if your tank is in any danger. To facilitate this, it is recommended to create at least two tasks to handle the motions of the tank and the tracking of missiles. This is not a requirement, and you may actually create any number of tasks you feel necessary to best implement this system.
- Some of the operations or math functions you decide to implement may be quite complex. You are allowed to write additional helper functions to take care of these for you, and you may organize your code any way you like as long as the **YourName_init()** function stays in **YourName_app.c**.