ECE 461/561 EMBEDDED SYSTEM DESIGN PROJECT 3: SPEED OPTIMIZATION OF WATER CURRENT CALCULATOR

OVERVIEW

For this project you will optimize the speed of code which calculates water current magnitude and direction based on the motion of a boat. If there is current, the boat's speed through the water will differ from its speed over ground, and the boat may not move in the direction it is pointed.

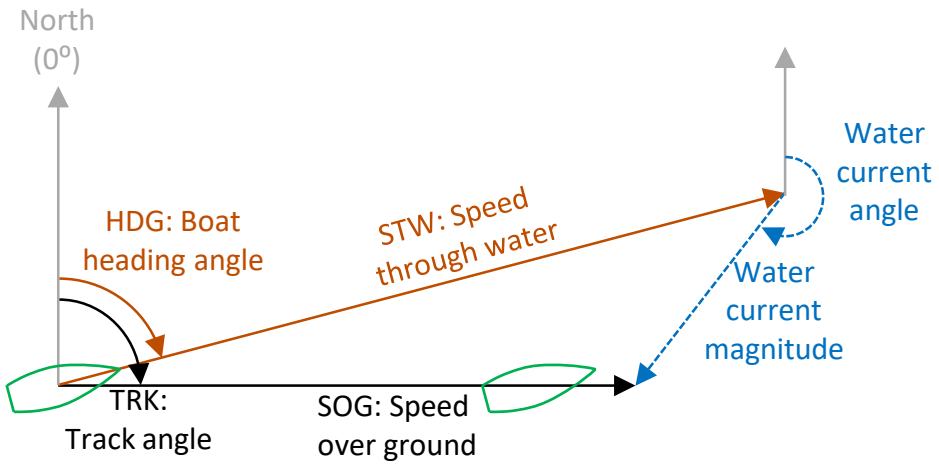

Figure 1. Terms associated with detecting drift in water due to current.

In Figure 2, the track is the direction of the path followed over the ground, while the heading is the direction the vehicle is pointing. Speed is measured in knots (nautical miles per hour, 1 kt ≈ 1.2 mph), and heading is measured in degrees from north (0°). This web site calculates current and direction: http://hyperphysics.phy-astr.gsu.edu/hbase/airpw2.html#c1

BASE PROGRAM DESCRIPTION

The starter code for this project is available at https://github.ncsu.edu/agdean/ESD/tree/master/Speed/Project_3_Base . The program has three phases:

1. Initialization of peripherals and data structures and other run-time support structures.
2. Calculation of current magnitude and direction for fixed test cases which are included in the program. The code includes comparisons with correct values to determine if your optimizations are sufficiently accurate.
3. Calculation of current magnitude and direction for test cases entered by serial port. This portion of the code will be used to grade your project by evaluating execution times for additional test cases.

The blue LED output (port D bit 1) is used to provide timing information visible on a logic analyzer. This will be used in grading.

PROFILING

The project has profiling support integrated into the project. Be sure to install the Cygwin and the profiling tools on your PC. Details are available on github at https://github.ncsu.edu/agdean/ESD/blob/master/Tools/ProfilingTools/Profiling%20Tools.pdf

RECEIVING INPUT DATA AND SENDING RESULTS

Serial communications are used to report on calculation accuracy and the execution time profile. The program communicates with the PC via UART0/Virtual Serial Port in the OpenSDA debug MCU. UART0 is configured for 115,200 baud, no parity, and one stop bit. Polling is used for transmission and reception.

The Segger J-Link OpenSDA debugger program supports both the debug and virtual serial port communications required for this project. Other debuggers may not.

REQUIREMENTS

GENERAL

1. You may complete this project individually (10% extra credit) or with one partner.
2. Your code may be examined for possible plagiarism by using MOSS (http://theory.stanford.edu/~aiken/moss/).

CODE MODIFICATIONS

1. Modify the base code to send your name and your NCSU Unity ID (and your partner's too, if applicable) upon start-up.
2. Reduce the execution time of the function Compute_Current(). The maximum acceptable errors for your optimized calculations are 0.1 knots for speed and 1^{o} for direction. The following optimizations are recommended:
   - Setting compiler to maximum optimization level for speed and enabling fast math.
   - Converting math operations from double-precision to single-precision.
   - Eliminating common sub-expressions and reusing values which were calculated previously.
   - Moving code to execute only if needed.
   - Using approximations for trigonometric and other functions. The file trig_approx.c includes approximation code which is described online here: http://www.ganssle.com/approx.htm
   - Simplifying range reduction in sin and cos approximations based on limited input data range.

DELIVERABLES

Submit the following through Moodle.

- Zipped archive of project.
- Report indicating profiles, optimizations and performance impacts.

GRADING

The following factors will be used to determine your grade.

- Output data correctness. The errors for each point must not exceed the values specified above.
- Execution speed for both the test cases in the code and an additional set of test cases. To get full performance credit, your code should be able to execute the test calculations (100 iterations of the 13 tests, with profiling disabled and no printfs or error checking) in the times below. For reference, the base without optimizations, profiling, serial communications or error checking code take about 340 ms.
  - ECE 461: <110 ms

  - ECE 561: <85 ms
- Report – see template for details.
- Code quality. The instructors may deduct points for remarkably bad coding practices (for example functions longer than a page, magic numbers, non-descriptive names). These are described online at http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html. If you have a question about what is acceptable, please post it on the class discussion forum so others can learn as well.