PS03: Multi-Threaded Robot Programming in C

Due: March 26, 2014

Robot systems need to sense, process, and act on information in real-time, with strict timing guarantees. For this assignment, you will experiment with the multi-threaded kernel on the rone robots, and then build two fun behaviors in C: Avoid obstacles, and wall-follow.

1 Threads, Mutexes, and Messages

Make a two-thread program based on `tbd.c`. Each thread should print a unique string 10 times. Something like, "Hippopotamus" and "Platypus". Do not use any yield or delay functions from the FreeRTOS API. Use `serial_send_string()` to print over the serial port.

1. Make thread 1 a higher priority than thread 2. Capture the output and hand in.
2. Make the two threads the same priority. Capture the output and hand in.
3. Make a new function, `serial_send_string_mutex()`. Use a mutex to ensure that only one thread can print at a time. Capture the output and hand in.

Make a three-thread program and a message queue. Thread 1 and 2 should put 10 total messages on the queue, one every 0.5 second. The messages should be pointers to the strings from above, and use a different string for each thread. Thread 3 should read the queue and print the message. You will need to read about how to implement periodic threads in the FreeRTOS book.

1. Make thread 1, 2, and 3 the same priority. Capture the output and hand in.

2 Obstacle Detection and Wall Folowing

Make a new program based on `tbd.c`. Use the background thread to read the obstacle detector with the `irRangeGetBits()` function.

1. Make a `obstacleAngleCompute()` function that takes the obstacle bits and computes the direction of the obstacle. Refer to the `process_nbr_message()` function for inspiration on computing direction from bits. Note that you will potentially need to deal with obstacles on many different sides of the robot.
2. Make a `avoidObstacles()` function that takes the obstacle angle (and maybe the bits, too) and steers the robot away from obstacles. Put this function into a programm to make the robot wander around the environment.
3. Make a `followWall()` function that takes the obstacle angle (and maybe the bits, too) and drives the robot along a wall. (Note: I'm only about 40% sure if this is possible...)

2.1 Hand-In / Check-Off

1. **Hand-in:** Your traces from Section 1.
2. **Check-off:** Your `avoidObstacles()` and `followWall()` function in operation.
3. **Hand-in:** Submit your code to Owl-Space.

2015-02-27