16.35 - Real-Time Systems and Software
Asst. #2

Handed Out: Feburary 27th, 2020
Predeliverable Due: March 5th, 2020 at 5:00 PM
Due: March 10th, 2020 at 5:00 PM

Submission Instructions

Please submit your assignments electronically using `git`, and upload the files to Gradescope. Some notes about the submission:

1. Please maintain the file structure and naming structure in the solution. Changing the name of a file may cause your work to not be graded and will likely annoy the course staff.
2. This assignment will make use of `cmake`. You do not require familiarity with `cmake` to complete the assignment.
3. The written section will require you to report the time that you spent on each question. Please bear this in mind when developing your solutions.
4. The predeliverable will consist of a fully completed question 1.

Please ensure that your code compiles and runs from within your source folder. You will not receive grades for code that does not compile.

Setup

This code, as written and provided, supports UNIX operating systems. Your assignments from here will make extensive use of the `pthread` package, known as POSIX-threads. POSIX-threads are not (natively) supported on Windows. We highly advise setting up and working with a virtual machine and/or a linux distribution for this and future assignments.

You are welcome to modify the code provided to work on Windows but note that no support will be given.

The code provided should not require the installation of additional packages (although a barebones installation may require you to install `cmake` and `python2.7`).

Compiling

The project provided will contain a file called `CMakeLists.txt`. `cmake` is a build system - it generates the makefiles that one would traditionally write, and provides high-level functionality to intelligently do so. `cmake` is extremely widely used; it is simple, exact and has a nice testing framework that we'll find useful in this assignment. Note that other build systems exist and have comparable capabilities - eg. gradle, autoconf, Premake, Ninja etc.

Please follow these steps for building the project:

1. Navigate into the directory containing the assignment files (you should see a file called `CMakeLists.txt`).
2. Create the build folder and change directory to it
```
$ mkdir build && cd build/
```

3. Now run `cmake`, with the root folder in the parent directory:

   ```
   $ cmake ..
   ```

4. `cmake` will generate the make file for you. To build the application, simply run

   ```
   $ make main
   ```

5. To run the application, simply call `./main` in the build directory:

   ```
   $ ./main
   ```

Assignment Overview

This assignment will require you to implement a specification, and then extend the specification to meet new design criteria (as a partially parallelized program using what you will cover in class). You will be provided with a code framework to get you started for the programming portion of this assignment. As part of implementing the specification, you will be required to implement several tests for code that you have written to ensure that it meets the specification. The code can be found on Learning Modules.

Partial credit will be given for incorrect implementations, provided that they **compile** (code that does not compile will receive no credit). The total number of available points is 145.

1 Implementing a Specification [Predeliverable - 50 Points]

In the last assignment, we asked you to think through the requirements for a *vehicle simulator*! In this assignment, we're going to ask you to implement a similar specification that we developed for a similar problem. We will assess your correct interpretation of the requirements that are necessary to implement; you will lose grades if you do not implement the **shall**s but you will not lose grades for not implementing the **should**s.

The specification is divided into sections for the vehicle and the controller. You will need to implement files `vehicle.c` and `controller.c` and provide a screenshot of the working system. The simulator and main functions have been written and provided for you.

1. Vehicle

   1.1. A vehicle shall be represented as a structure of name `t_vehicle` and type `vehicle`, with the following properties:

name	**signature**	**description**
`position`	`double *`	$[x, y, \theta]$
`velocity`	`double *`	$[\dot{x}, \dot{y}, \dot{\theta}]$
`num_waypoints`	`int`	the number of waypoints
`target_waypoints`	`double **`	the set of available waypoints
`current_waypoint`	`double *`	the pointer to the current waypoints (in `target_waypoint`)
`current_waypoint_idx`	`int`	the index of the current waypoint
`set_position`	`<function-ptr>`	function pointer to set vehicle position
`set_velocity`	`<function-ptr>`	function pointer to set vehicle velocity
`control_vehicle`	`<function-ptr>`	function pointer to control the vehicle
`update_state`	`<function-ptr>`	function pointer to update vehicle states

Table 1: Specification for structure `t_vehicle`

   1.2. Vehicle positions x and y shall not exceed the range $[0, 100)$.

1.3. Vehicle heading, θ, shall be limited to the range $[-\pi, \pi)$.

1.4. Vehicle linear velocity in the x and y plane shall not exceed the range $[5, 10]$.

1.5. Vehicle angular velocity shall not exceed the range $[-\pi/4, \pi/4]$.

1.6. All waypoints (represented as x and y) shall fall within permissible vehicle positions and shall be unique.

1.7. `target_waypoints` shall contain `num_waypoints` elements, representing waypoints.

1.8. The `current_waypoint_idx`$^{\text{th}}$ element of `target_waypoints` shall hold the internal representation of `current_-waypoint`.

1.9. The function pointers in Table 1 shall map to functions providing the following functionality:

1.9.1 `set_position` shall take the form `void set_position(struct t_vehicle *v, double * values)` and shall set the position of the vehicle subject to the limitations outlined in 1.2 and 1.3.

1.9.2 `set_velocity` shall take the form `void set_velocity (struct t_vehicle *v,double * values)` and shall set the velocity of the vehicle subject to the limitations outlined in 1.4 and 1.5.

1.9.3 `control_vehicle` shall take the form `void control_vehicle(struct t_vehicle * v)` and shall be set in the initialization routine to a function that calls `get_proportional_waypoint_control` (the vehicle waypoint tracking controller) and applies the generated control.

1.9.4 `update_state` shall take the form `void update_state(struct t_vehicle * v, double delta_t)` and update the state of the vehicle based on the dynamics in Equations 1 to 3:

$$x = x + \dot{x} \cdot \delta t \tag{1}$$

$$y = x + \dot{y} \cdot \delta t \tag{2}$$

$$\theta = \theta + \dot{\theta} \cdot \delta t \tag{3}$$

1.9.5 `set_velocity` should take gravitational warping into account.

1.10. A vehicle shall be created through a function with signature
`vehicle * create_vehicle(double * starting_pos, int n_waypoints, double ** offset_waypoints)`
This function shall allocate memory required for the vehicle, initialize the vehicles position and waypoints subject to the constraints, and assign relevant functions to the function pointer variables in the structure in line with this specification.

2. Controller

2.1. A control shall be represented as a structure of name `t_control` and type `control`, with the following properties:

name	**signature**	**description**
`speed`	`double`	the forward speed of the vehicle.
`angular_velocity`	`double`	the angular velocity of the vehicle

Table 2: Specification for structure `t_control`

2.2. A controller shall include a single method of the form:
`control get_proportional_waypoint_control(struct t_vehicle * vehicle)`

2.3. `get_proportional_waypoint_control` shall accept as input a pointer to an object of type `t_vehicle` named `vehicle`. The `vehicle` structure shall obey the specifications for Vehicle objects, as outlined in 1.1-1.10 of the specification.

2.4. `get_proportional_waypoint_control` shall return a control object with a linear speed that obeys the Vehicle specification for linear speeds.

2.5. `get_proportional_waypoint_control` shall return a control object with an angular velocity that obeys the Vehicle specification for angular velocity.

2.6. Within the bounds of acceptable vehicle angular velocities, `get_proportional_waypoint_control` shall return a control object with an angular velocity equal to the difference in radians from the angle between the vehicle and its `current_waypoint` and the current angular heading of the vehicle.

2.7. The controller shall always operate on the smallest angle difference ($|\theta - \hat{\theta}| \leq \pi$).

2.8. The absolute control output shall be bounded by vehicle specifications 1.4 and 1.5.

2.9. A controller should provide an alternate control policy that, in polynomial time, finds the minimum-distance path that causes a vehicle to visit all target waypoints.

As you implement the above specification, we recommend you actively test your code either manually (by running the simulation) or through automated tests (as described in the next section). If you are testing manually, it helps to know the rough layout of this project.

Please note that the program communicates with a simple graphics program called the DisplayServer via a TCP socket. The DisplayServer runs in `python2.7` and will display the vehicles moving about the screen. You are welcome to modify, disable, or otherwise play with the DisplayServer as you wish - you will not be graded on it.

Please provide a screenshot of the vehicles following the waypoints in the DisplayServer with your written submission when submitting your final work to GradeScope

In later parts of this pset, you will modify the `run()` method.

Learning Objectives: Following a specification, identifying implementable properties of specifications, c programming

2 Unit Testing [Final - 25 Points]

One of the benefits of using the `cmake` build system is its built-in support for unit testing. In this section, we're going to implement a few unit tests. A unit test, as discussed in class, is a very simple program that calls functionality and verifies that it is correct. `cmake`'s test framework will run all your tests for you and identify the ones that failed.

To do so, you will create a separate executable in the file `CMakeLists.txt`. The following lines do exactly that:

```
add_executable(test_controller tests/test_controller.c ${SimulatorFiles})
target_link_libraries(test_controller m) # c's math library
```

You then define a test where the appropriate target is called (the name of the executable in this case), with whatever arguments you choose to pass to it:

```
add_test(TestController1 test_controller fail) # should fail
add_test(TestController2 test_controller pass) # should pass
```

Tests are deemed to have passed based on the return value of the executable. See the file `tests/test_controller.c` for details. Ensure that your executables build and are built (use `$ make all`). Running tests is then very simple:

```
$ make test
```

will produce results along the lines of

```
Running tests...
Test project .../build
    Start 1: TestController1
1/2 Test #1: TestController1 ..................***Failed    0.00 sec
    Start 2: TestController2
2/2 Test #2: TestController2 ..................   Passed    0.00 sec
```

```
50% tests passed, 1 tests failed out of 2

Total Test time (real) =   0.00 sec

The following tests FAILED:
  1 - TestController1 (Failed)
Errors while running CTest
```

Please create appropriate unit tests for the following elements of your code:

1. The control value returned by your program is consistent given three different starting locations and first waypoints.
2. Updating a vehicle state maintains the state within the position and velocity bounds as stated in the specification.

You are not limited in the number of tests that you may run. Please ensure that all your tests pass and operate as expected.

Learning Objectives: Unit testing, requirements testing, ctest, rudimentary cmake

3 Basic Threading [Final - 25 Points]

In the simulator's run method, each vehicle currently calls its controller and updates its states:

```
while (sim->current_time < sim->max_time) {
    // other code
    // ...
    for (vehicle * v = sim->vehicles; v < sim->vehicles + sim->n_vehicles; v++){
        v->control_vehicle(v);
        v->update_state(v,sim->time_increment);
    }
    usleep(sim->time_increment*1e6);
}
```

In this section, we will be exploring some basic threading. We want to see how long this process takes to execute *before* we attempt to thread it, and then see how long it takes after we have parallelized this process.

Your task in this section is to do the following:

1. Modify the default simulator `run()` method to, over the course of your simulation, record the **average** time that each iteration of the while loop takes to update *without threading*. The method should be modified in `simulator.c` and included in your final submission.
2. Implement a second run method called `run_threaded()` in which each vehicle is updated on a separate thread (the rest of the method may be identical). In the method, the threads shall be created and joined *in place of the vehicle for loop* (i.e. the threads shall be created and destroyed in each iteration of the while loop). The method should be implemented in `simulator.c` and included in your final submission.
3. Modify your program to accept a run time parameter that causes the program to use the appropriate simulator run method. For example, passing `--threaded` could cause it to run a threaded execution, while omitting any input to the program could have it run the default method. Make sure to add comments to indicate what parameters you expect and how they affect your code.

4. Report the **average** time that each iteration of the while loop in `run_threaded()` takes.

In your report, please answer the following questions:

1. How did the threaded version compare to the non-threaded version?
2. How do you believe performance will scale with the number of vehicles? Will threading become more appropriate as we have more vehicles?
3. Can this threading process be made more efficient?
4. Is this thread-safe?

Learning Objectives: Basic thread control, when to thread

4 Writing a Thread-safe Specification [Final - 25 Points]

The code, as is, is not particularly efficient and does not make great use of the threading resources. Let us consider instead, a situation in which each vehicle, and the simulator, exist on separate threads.

Please create a specification for a program threaded in such a way and describe its operation. You may use condition variables and mutexes for controlling the thread execution. You do not need to specify any part of your program in the pthread syntax (you may if you so choose), but you *must* identify the shared critical regions and which objects hold which mutexes and where condition variables are used.

Please list the termination conditions of your specification, and explain (in words) how thread execution control will ensure that your program will terminate and tick appropriately.

DO NOT IMPLEMENT THIS SPECIFICATION

Learning Objectives: Specification writing, threading implications, thread execution control

5 Git Log Message [Final - 10 Points]

Now that you have some experience with git, we would like to see good programming practices on your part. As part of your write up, please include a git commit message and log entry. For example, the following would be sufficient for this section (although ideally you would write something more in the commit message).

```
commit c18da58dd00a474d3d07203200ca69c3a95b670a (HEAD -> master)
Author: Ace Ventura <aceventura@petdetectivellc.com>
Date:   Sun Mar 12 13:27:08 2019 -0500

    Completed assignment 1.
```

Learning Objectives: Basic git, good programming practices

6 Report the Time that you Spent on Each Question [Final - 10 Points]

Please report the time spent on Questions 1 - 5.

Learning Objectives: Time tracking for billable hours (standard development practice)