16.35 PSet #2

Ryan Fish

February 26, 2015

1 VehicleController

2 GroundVehicle

2.1 Requirements

2.1.1 Variables

1. `pose[]` shall be an array of exactly 3 elements that determine the position and orientation of a vehicle on a 2D plane.
   (a) `pose[0]`, which represents the x-coordinate of said vehicle, shall remain at all times within the range $[0, 100]$.
   (b) `pose[1]`, which represents the y-coordinate of said vehicle, shall remain at all times within the range $[0, 100]$.
   (c) `pose[2]`, which represents the yaw of said vehicle, shall remain at all times within the range $[-\pi, \pi)$.
      i. If the vehicle would otherwise be set to a value outside of these bounds, the value shall be wrapped to the equivalent angle within the range.
2. `speeds[]` shall be an array of exactly 2 elements that determine the derivative of position and orientation.
   (a) `speeds[0]`, which represents the absolute linear velocity of said vehicle, shall remain at all times within the range $[3, 10]$.
      i. In order to maintain a no-slip condition, the linear velocity shall be described only as a magnitude, with the variable described by `1(c)` determining the direction of motion.
   (b) `speeds[1]`, which represents the rotational velocity of said vehicle, shall remain at all times within the range $[-\pi/4, \pi/4]$.

2.1.2 Construction

1. There shall be one constructor for the `GroundVehicle` object. The constructor shall take 3 inputs.
   (a) The first input shall be an array defining the vehicle starting position as defined in 2.1.1.1
   (b) The second input shall be a value defining the vehicle linear velocity as defined in 2.1.1.2(a)
   (c) The third input shall be a value defining the vehicle rotational velocity as defined in 2.1.1.2(b)
2. If any inputs are invalid in range as described by 2.1.1.1 and 2.1.1.2, but valid in type, the constructor shall set the inputs to their nearest legal bounds and result in a valid object being returned.
3. If the inputs to the constructor are invalid in number or type, the constructor shall throw an error signifying invalid inputs.

2.1.3 Methods

1. `getPosition`
   (a) The method shall take nothing as input.
   (b) The method shall return, as an array, the vehicle position as defined in 2.1.1.1
2. `getVelocity`
   (a) The method shall take nothing as input.
   (b) The method shall return, as an array with three values, the vehicle velocities.
      i. The first element of the returned array shall be the x component of the vehicle's velocity in the coordinate frame from which the vehicle orientation is referenced.
      ii. The second element of the returned array shall be the y component of the vehicle's velocity in the coordinate frame from which the vehicle orientation is referenced.
      iii. The third element of the returned array shall be the rotational component of the vehicle's velocity.
3. `setPosition`
   (a) The method shall take as input an array as defined in 2.1.1.1
   (b) The method shall validate the inputs in a manner consistent with 2.1.2.2 and 2.1.2.3
   (c) The method shall set the vehicle `pose[]` variable to the input array.
4. `setVelocity`
   (a) The method shall take as input an array as defined in 2.1.3.2(b)
   (b) The method shall validate the inputs in a manner consistent with 2.1.2.2 and 2.1.2.3
   (c) The method shall set the linear velocity as defined in 2.1.1.2(a) as the euclidean norm of the first two input array elements.
      i. To ensure consistency of the no-slip condition, the method shall also set the vehicle heading as defined in 2.1.1.1(c) to that of the vector resultant of the sums of the input x and y velocities.
   (d) The method shall set the rotational velocity as defined in 2.1.1.2(b) to the value of the third array element.
5. `controlVehicle`
   (a) The method shall take as input a single instance of a `Control` object.
   (b) The method shall validate the input in a manner consistent with 2.1.2.3
   (c) The method shall return immediately upon receiving a null `Control` object.
   (d) The method shall validate the velocity fields of the `Control` object in a manner consistent with 2.1.1.2
   (e) The method shall set the linear and rotational velocity fields as defined in 2.1.1.2 of the vehicle to the values stored in the respective fields of the `Control` object.
6. `updateState`
   (a) The method shall take as input two arguments
      i. An unsigned integer representing the time since the vehicle started in seconds, floored.
      ii. An unsigned integer representing the additional milliseconds elapsed since the time given in the first argument.

iii. If both inputs are 0, the method shall preemptively exit.

iv. If either input is signed, the method shall exit, throwing an error signifying invalid inputs.

(b) The method shall validate the input in a manner consistent with 2.1.2.3

(c) The method shall set the new position of the vehicle based on the time elapsed and the vehicle speeds as of the update.

i. The final position shall not exceed the limits stated in 2.1.1.1

ii. The delta between the current and final position shall be approximated as a straight line in the case that the absolute value of the rotational velocity as defined in 2.1.1.2b is less than $s * 2 * \pi * MIN\_VALUE$, where `MIN_VALUE` is defined as the value closest to 0 of the data primitive used.

iii. If the absolute value of the rotational velocity exceeds that specified in 2.1.3.6(c)ii, the exact, partial-arc solution shall be used in calculating the new position. The $2 * \pi$ divided by the rotational velocity determines the time the vehicle would take to rotate through a full circle. This time multiplied by the linear velocity determines the circumference of that circle, and therefore its radius and diameter. The new position of the vehicle is given by the arc segment given by the update duration times the rotational velocity, of the circle tangent to the vehicle's current vehicle with the radius previously calculated.

3 Simulator

4 Mutex and Sync

4.1 Shared Resources

1. Each `GroundVehicle` is shared between threads
2. The time is controlled from `Simulator`, but read from many.
3. The `GroundVehicle` position is controlled from one thread, but read from many.
4. The status variable indicating all `VehicleControllers` have completed needs to be read and written by many.

4.2 Critical Regions

1. Modifying the state of the `GroundVehicle`, as in, changing velocity or position, should be done from one and only one thread at a time.
2. Modifying time must be an atomic operation, so listening threads never read a partially updated variable.
3. The position output to the client is managed by one thread, but many threads have updated positions. Changing the pose of a vehicle must be atomic, and the output to the client shall be managed by a single thread, using a buffer that has atomic write operations.
4. Incrementing the status variable to indicate thread cycle completion shall be an atomic operation, as well as locking, to protect from READ-READ-MODIFY-WRITE-MODIFY-WRITE style errors.