SYSC-5709–W

Software Programming in C

Carleton
UNIVERSITY

SOFTWARE CONTROLLER FOR VEHICLES

Github Repository: https://github.com/PCoser/Software-Programing-in-C

Group members

- Pedro Coser
  ID: 101126424
  pedrocoser@cmail.carleton.ca
  GITHUB ID : PCoser

- Unnati Thakkar
  ID: 101164481
  unnatithakkar@cmail.carleton.ca
  GITHUB ID : unnati9979

a. Problem Statement

Thinking about the development of a new series of vehicles, an automobile company reached us out in order to ask our services for designing the software controller and the integration with all the devices of the vehicles. This series will be designed with the concept of All Drive-by-Wire technology, meaning that there will not be any mechanical linkage between the devices, all commands from the driver will be sent through communication protocol. Moreover, some autonomous functionality can also be added.

The client also requested us to develop a simulator to test the system without the real vehicle. The project is consisted of two routines one for the system we are developing and another for the simulator we will built to test the system. Both routines will communicate through a structure of data, this structure represents the complete status of the vehicle. Every function inside the routines will be allowed to read/write or both in the structure in the places where it is designed for.

The simulator routine will simulate the real behaviour of the vehicle. It will the status of the vehicle and update it accordingly to the dynamics of a real vehicle. The sensors reads from the interface as it was measuring the real status. The actuators write in the interface as it was acting in real devices.

The project includes the software of the central controller, the interface of the sensors of the wheels, steering wheel, brake and gas pedal, motor, gear and brake. Some details follow:

- Central Controller: Computer for control of the communication between devices, check the health of the system, perform automation as it needs.

- Steering wheel sensor: read status from the interface representing the Steering wheel and put this information in the protocol when it is requested by the controller.
- Pedal throttle sensor: read status from the interface representing the Pedal throttle and put this information in the protocol when it is requested by the controller.
- Pedal gas sensor: read status from the interface representing the Pedal gas and put this information in the protocol when it is requested by the controller.
- 4x Wheel sensor: read status from the interface representing the wheels and put this information in the protocol when it is requested by the controller.
- Range Sensor: reads the measurement from the interface representing the range(distance from the vehicle ahead) and put this information in the protocol.
- Lines sensor: reads the measurement of the distance to the line and the orientation from the interface representing the vision system and put this information in the protocol.
- Motor actuator: reads the information of the protocol, perform the output to the hardware and return a confirmation signal to the controller.
- Brake actuator: reads the information of the protocol, perform the output to the hardware and return a confirmation signal to the controller.
- Direction actuator: reads the information of the protocol, perform the output to the hardware and return a confirmation signal to the controller.

The control of the motor and the gears is a complex system. We are assuming that the vehicle is single gear. So we no need to deal with this issue. Indeed, it is out of the scope of the project.

b. Requirements

The software should be built to meet the clients needs. Some client's need concerns to the functionalities of the systems and this is included in the first release:

1. Throttle, break, steer, shift commands from the driver should be read by the sensors and executed by the central controller or other devices.
2. The simulator needs to perform as it was a real vehicle, interfacing with the system through the structure data representing the vehicle status.
3. The simulator should be able to receive a set of inputs representing the actions performed by a driver and pass this information to the vehicle status.

Other needs in this industry is concerned to the reliability of the system and will be part of the second release:

4. The central controller should be able to communicate with every device of the system in real-time. For that, the client requires a watchdog function, where the central controller can track the time between requests and replies.
5. The devices should be able to return a fault message if the command could not be executed.
6. The devices should periodically send a "life" signal to the central controller to ensure it is "alive".

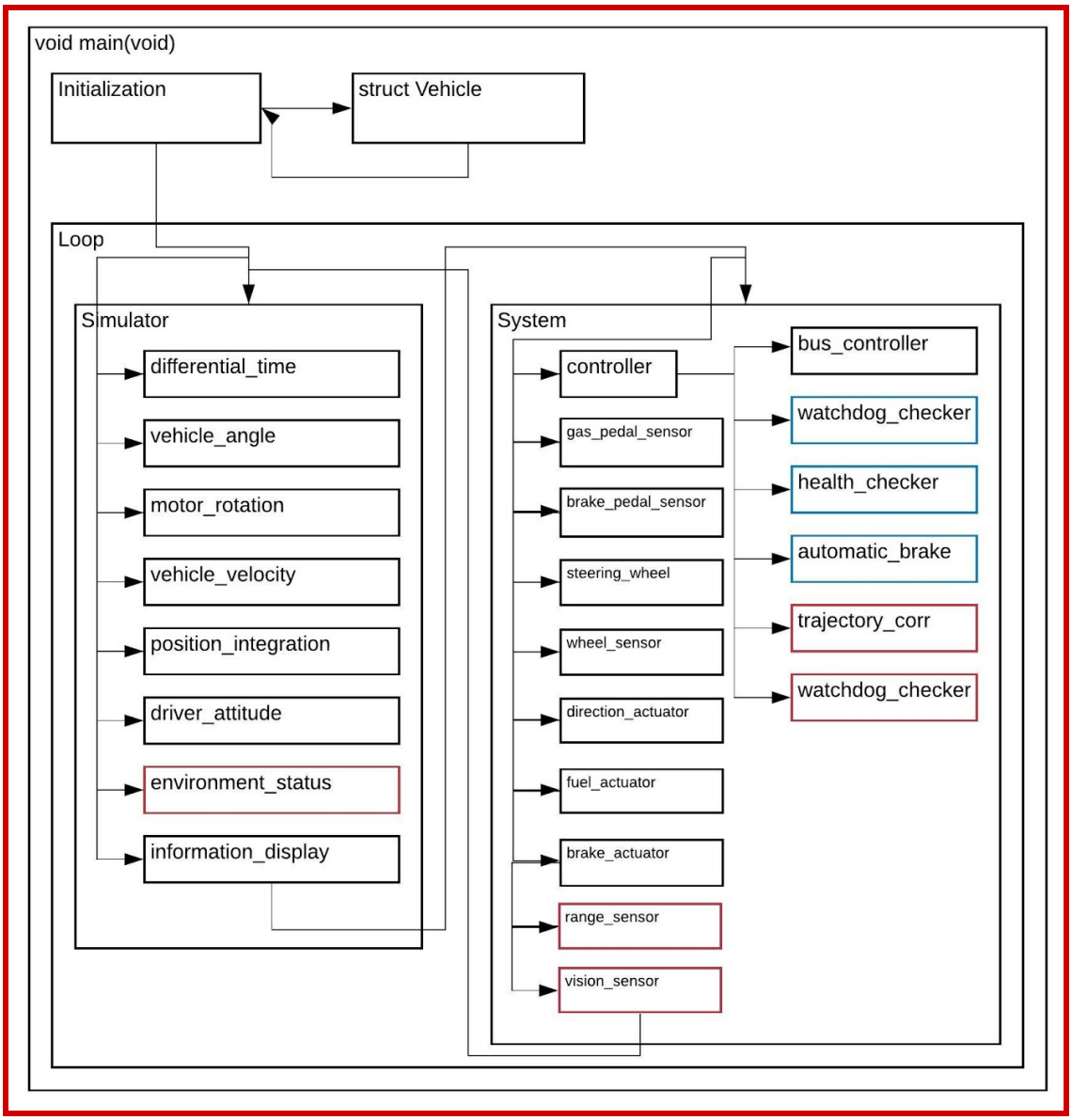
The client put some functionality to be added latter, this is part of the third release:

7. The central controller should execute an autonomous break according to the information from a front range sensor.
8. The central controller should execute an autonomous correction in the trajectory and alert the driver if, from the information provided by a front vision system, the car hit the edge of the lane.

Summarizing, the first release will cover the basic functionalities; the second release will cover the reliability requirements; the final release will cover additional functionalities concerned to autonomous driving.

c. Software Design

The implementation consists in two routines inside a loop.
The loop last until all commands from the driver were performed.
The simulation routine calls all its functions sequentially.
The system routine calls the functions sequentially. Except by the functions that is called inside the function controller, that is called as it is needed.
The functions in blue are for the second release, the functions in red for the third release, all others in black is for the first release.



struct Vehicle

The interface between the two routines represents physical features of the vehicle to be controlled by both. It will be a structure of data containing the following informations:

```
double vehicle_wheel_angle; // Range()
double vehicle_wheel_rotation; // Range()
double vehicle_position_X;
double vehicle_position_Y;
double motor_rotation;
double gas_pedal_pos; // Range(0-100)
double brake_pedal_pos; // Range(0-100)
double steering_wheel_pos; // Range(-100,100)
double direction_actuator_pos; // Range(-100,100)
double fuel_actuator_pos; // Range(0-100)
double brake_actuator_pos; // Range(0-100)
double range_sensor_val;
double vision_system_val;
int Comm_bus_address;
float Comm_bus_message;
```

List of functions:

- void main(void) - Pedro

Simulator:

- differential_time; - Unnati
  description: This function computes the time passed since the last call, it is used to compute the equations that simulates the vehicle.
  input: system clock
  output: differential time
  parameters: void
  return: differential time
  release: 1

- vehicle_angle; - Unnati
  description: Computes the angle of the vehicle in relation to a reference frame.
  input: wheels attitude, velocity, differential time
  output: vehicle angle
  parameters: differential time
  return: void
  release: 1

- vehicle_velocity; - Unnati
  description: calculates the speed of the vehicle based on the power of the motor.
  input: power of the motor
  output: vehicle speed
  parameters: differential time
  return: void
  release: 1

- position_integration; - Unnati
  description: Integrates the velocity into position
  input: vehicle angle, vehicle velocity, differential time
  output: position (x,y)
  parameters: differential time
  return: void
  release: 1

- driver_attitude; - Pedro
  description: This function aims to simulate the driver's attitude driving the vehicle. So, it will convert a given trajectory into actions in the steering wheel, gas and brake pedal.
  input: file containing the trajectory
  output: Steering wheel, gas pedal, brake pedal.
  parameters: time sampling, system time
  return: void
  release: 1

- environment_status; - Pedro
  description: This function aims to simulate external inputs to the system, such as: any vehicle ahead or trajectory disturbances. This is needed in order to implement the automatic actions of the controller.
  input: file containing the environment status
  output: range sensor, lane sensor
  parameters: system time
  return: void
  release: 3

- information_display; - Pedro
  description: This shows the status of the vehicle in runtime.
  input: vehicle status
  output: display, csv file
  parameters: system time
  return: void
  release: 1

System

- controller - Pedro
  description: This function plays the role of the Central Controller of the vehicle. It calls other subroutines and pass information through the communication bus. It also keeps track of the health of the devices
  input: Information from the bus
  output: Commands to devices.
  parameters: bus message
  return: void
  release: 1

  - bus_controller - Pedro
    description: This is a function to be called inside the controller. It controls the information traffic between devices;
    input: Information from the bus.
    output: Commands to devices.
    parameters: bus message
    return: int last_bus_address
    release: 1

  - watchdog_checker; - Pedro
    description: It check the time between requests from the controllers to the devices and replies from the device. If the time extrapolate a give parameter, it assumes that the bus crashed.
    input: System time, command from controller and replies **from** devices.
    output: alarm to the controller
    parameters: command time, command
    return: system alarm
    release: 2

  - health_checker; - Pedro
    description: check if every device is alive sending a standard command to the device and checking the reply.
    input: System time
    output: Alarm
    parameters: void
    return: health status
    release: 2

  - automatic_brake; - Unnati
    description: It perform an automatic brake is it is detected any danger ahead. It will send a brake command to the brake actuator
    input: range sensor measurement, vehicle speed
    output: command to device
    parameters: void
    return: brake position increase
    release: 3

- trajectory_corr; - Pedro
  description:It performs an automatic change in trajectory if the vehicle is going out of the lane.
  input: vision system measurement
  output: command to device
  parameters: void
  return: trajectory correction
  release: 3

- gas_pedal_sensor; - Pedro
  description: It plays the role of the gas pedal sensor measuring from the vehicle status the action in the gas pedal and passing this information to the controller.
  input: gas pedal position from the system status
  output: information to the controller
  parameters: bus message
  return: void
  release: 1

- brake_pedal_sensor - Pedro
  description: It plays the role of the brake pedal sensor measuring from the vehicle status the action in the brake pedal and passing this information to the controller.
  input: brake pedal position from the vehicle status
  output: information to the controller
  parameters: bus message
  return: void
  release: 1

- steering_wheel_sensor - Pedro
  description: It plays the role of the steering wheel sensor measuring from the vehicle status the action in the steering wheel and passing this information to the controller.
  input: Steering wheel position from the vehicle status
  output: information to the controller
  parameters: bus message
  return: void
  release: 1

- wheel_sensor - Pedro
  description: It plays the role of the rotation sensor measuring from the vehicle status the speed of the wheel and passing this information to the controller.
  input: rotation speed from the vehicle status
  output: information to the controller
  parameters: bus message
  return: void
  release: 1

- direction_actuator - Pedro
  description: It plays the role of the direction actuator to the front wheels, writing in the vehicle status the new desired angle of the wheels
  input: command from the controller
  output: confirmation to the controller
  parameters: bus message
  return: void
  release: 1

- fuel_actuator - Pedro
  description: It plays the role of the fuel actuator to the motor, writing in the vehicle status the new desired fuel intake to the motor.
  input: command from the controller
  output: confirmation to the controller
  parameters: bus message
  return: void
  release: 1

- brake_actuator - Pedro
  description: It plays the role of the brake actuator to the front wheels, writing in the vehicle status the new desired action in the pads of the brake.
  input: command from the controller
  output: confirmation to the controller
  parameters: bus message
  return: void
  release: 1

- range_sensor - Pedro
  description: It plays the role of the range sensor, measuring from the vehicle status the distance from any object ahead and passing this information to the controller.
  input: distance to the object ahead from the system status
  output: information to the controller
  parameters: bus message
  return: void
  release: 3

- vision_sensor - Pedro
  description: It plays the role of the vision system, measuring from the vehicle status the location of the vehicle in the lane and passing this information to the controller.
  input: location of the vehicle from the vehicle status
  output: information to the controller
  parameters: bus message
  return: bus message
  release: 3