ON-BOARD SPACECRAFT SOFTWARE

PROJECT 1 (Part A)
Basic task programming and unit testing

Tutor: Javier Fernández Muñoz

(jfmunoz@inf.uc3m.es)

Description

The objective is to build a prototype for reduced onboard software for a satellite. The system has two parts:

1. A main computing hardware system that controls the main tasks.
2. A microcontroller subsystem that has a direct access to the sensors and actuators.

Both subsystems communicate using a master/slave message protocol defined for this project.

1 Characteristics of the microcontroller subsystem:

The microcontroller is used for handling the hardware sensors and actuators while executing the commands received from the main system. The main tasks that this subsystem should perform are the following:

- Get the average temperature of the satellite. (This task is going be simulated instead of using a real sensor).
- Get the current position of the satellite on the orbit. (This task is going be simulated instead of using a real sensor).
- Receive commands from the main system, execute them and send back the response.

Also there are a few minor tasks that this subsystem should do and will be implemented on the next iteration of the project:

- Get the status of the sunlight onto the satellite. (This task is going to use a light sensor).
- Set the satellite heater on and off (This task is going to use a led as the heater and will be done on the next part of the project).

The hardware for this subsystem will be implemented (in the next iteration of the project) using an Arduino microcontroller with the necessary sensors and actuators.

In this iteration, the source code for the main task of this microcontroller subsystem is contained in two source files. This source files are:

- ***arduino_code.c***: This is the source file where each of the mentioned tasks should be implemented with one function. Also the state of the systems is stored using global variables.
- ***arduino_code.h***: This is the source file where the interface of these tasks is exported for testing and other purposes. Also some of the variables of the systems state are also exported to perform the tests.

Apart from the source code files, a test source file is also needed to perform the unit tests. The unit tests require the Google test platform to execute. The files are:

- ***test_arduino_code.cpp***: This is the source file for all the unit tests implemented for the Arduino microcontroller subsystem.

2 Description of the tasks: Get the average temperature of the satellite.

The task designed to obtain the temperature of the satellite is a simulated task that requires handling some of the main state variables of the system. The variables required are:

- ***heater_on***: This is an integer variable that acts as a boolean state defining whether the heater is on or off at this moment.
- ***sunlight_on***: This is an integer variable that acts as a boolean state defining whether the sunlight is hitting on the satellite right now.
- ***temperature***: This is a double variable that states the actual temperature of the satellite.
- ***time_temperature***: This is a double variable that states the date and time when the last temperature of the satellite was taken.

Also some constant values are also required as follow:

- ***SHIP_SPECIFIC_HEAT***: This is a double representing the specific heat of the satellite.
- ***SHIP_MASS***: This is a double representing the mass (in Kg) of the satellite.
- ***HEATER_POWER***: This is a double representing the energy that the heater can emit every second.
- ***SUNLIGHT_POWER***: This is a double representing the energy that the sun can emit towards the satellite every second.
- ***HEAT_POWER_LOSS***: This is a double representing the energy that the satellite loses every second.

To get the current time, an auxiliary function is provided, ***getClock(),*** that returns a double containing the actual time since EPOCH measured in milliseconds.

The algorithm for this task is the following:

- First, take the elapsed time since the last computation of the temperature.
- Second, calculate the total power gained or lost by the satellite.
  - Add the heat loss power every time.
  - If the heater is on, add the heater power.
  - If the sunlight is on the satellite, add the sunlight power.
- Then, the energy transferred is calculated multiplying the total power by the elapsed time since the last calculation.
- Finally the new temperature must be obtained using this formula:
  - NEW_TEMP = (ENERGY / (SPECIFIC_HEAT*MASS)) + OLD_TEMP
- Do not forget to update the state with the new temperature and the new temperature time

3 Description of the tasks: Get the current position of the satellite on the orbit.

The task designed to obtain the current position of the satellite is a simulated task that uses an array with a set of orbit positions. The task computes the actual position by interpolating between the two nearest positions from the array. This requires handling some of the main state variables of the system. The variables required are:

- ***position***: This is a ***struct position*** variable that stores the actual position of the satellite.
- ***init_time_orbit***: This is a double variable that states the date and time when the first orbit of the satellite started.
- ***orbit_points***: This is an array of ***struct position*** variables That stores a collection of orbit position evenly separated. This array defines the orbit.

Also some constant values are also required as follow:

- ***ORBIT_POINTS_SIZE***: This is an integer representing the size of the array of positions defining the orbit.
- ***ORBIT_TIME***: This is a double representing the time to travel a complete orbit.

To get the current time an auxiliary function is provided ***getClock()*** that returns a double that contains the actual time since EPOCH measured in milliseconds.

The algorithm for this task is the following:

- First, calculate the time elapsed since the last orbit started (relative time).
  - Subtract the actual time and the initial time of the first orbit. Then get the division module of the result and the time for a single orbit.

- Second, calculate the two consecutive index of the orbit positions array that surrounds the actual relative time, as following:
  - The previous position index is obtained as following:
    - Multiply the relative time (obtained before) by the total size of the positions array
    - Divide the result by the total time for one orbit.
  - The next position index is obtained as following:
    - Increment (add one) the previous position index
    - Calculate the division module of the result by total size of the positions array.

- Third, calculate the offset ratio of the actual position as following:
  - offset_ratio = (relative_time / (ORBIT_TIME / ORBIT_POINTS_SIZE)) -
    - previous_position_index;

- Finally, calculate each position coordinate by interpolating as follows:
  - new_coord = previous_position_index.coord * (1 - offset_ratio) +
    next_position_index * offset_ratio

  Do not forget to update the new position on the state.

4 Description of the tasks: Receive commands from the main system.

The task is designed to read the commands received by the main system, update the state and build the response message to be sent. The commands are stated as an enum type and are the following:

- ***NO_CMD***: No command has been received.
- ***SET_HEAT_CMD***: This command is used to setup the heater.
    - A boolean (***set_heater***) is needed to set the heater (on/off).
- ***READ_SUN_CMD***: This command is used to read the sun light sensor.
- ***READ_TEMP_CMD***: This command is used to read the temperature.
- ***READ_POS_CMD***: This command is used to read the position.

Each command has a corresponding message response, as following:

- ***NO_CMD***: This message has no response.
- ***SET_HEAT_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean)
- ***READ_SUN_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean)
    - ***data.sunlight_on***: returns the value of the sunlight sensor.
- ***READ_TEMP_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean).
    - ***data.temperature***: returns the value of the temperature.
- ***READ_POS_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean)
    - ***data.position***: returns the value of the position.

There are two global variables that stores the last received command and the ready-to-send response:

- ***last_cmd_msg***: This is an ***struct cmd_msg*** variable that contains the last received command.
- ***next_res_msg***: This is an ***struct res_msg*** variable that contains the ready-to-send response message.
-

The algorithm for this task is the following:

- First, analyze the last received command.
- second, update the state of the subsystem.
- Third, store the next response message in the corresponding global variable.
- Finally, set the last received command variable to NO_CMD.

5 Characteristics of the main computing subsystem:

The main computing subsystem is used for controlling the tasks and the interface with the user. The tasks that this subsystem should perform are the following:

- Control the temperature by selecting when to activate the heater. (Using the information from the microcontroller).
- Prepare to send a selected command from the list to the Arduino system,
- Store a received response from the Arduino to the last sent command.

Also there are a few minor tasks that this subsystem should do and will be implemented on the last iteration of the project:

- Print the status of the satellite periodically.
- Handle the following actuator and sensor by sending the request and receiving the proper response.
    - Heater activation.
    - Sun light sensor reading.
    - Temperature sensor reading.
    - Position sensor reading.

  These procedures will need the tasks 'prepare-to-send' and 'store-a-received' developed in this iteration.

The hardware for this subsystem will be implemented in the last part of the project using a i386 connected to the Arduino.

The source code required for this i386 subsystem should be contained in two source files. This source files are:

- ***i386_code.c***: This is the source file where each one of the tasks should be implemented with a single function. Also the state of the systems is stored using global variables.
- ***i386_code.h***: This is the source file where the interface of these tasks is exported for testing and other purposes. Also some of the variables of the systems state are also exported to perform the tests.

Apart from the source code files, a test source file is also needed to perform the unit tests. The unit tests require the Google test platform to execute. The files are:

- ***test_i386_code.cpp***: This is the source file for all the unit tests implemented for the i386 microcontroller subsystem.

6 Description of the tasks: Control the temperature by selecting when to activate the heater.

The task designed to select when to activate the heater is a control task that requires handling some of the main state variables of the system. The variables required are:

- ***temperature***: This is an double variable that states the actual temperature of the satellite.
- ***heater_on***: This is an integer variable that acts as a boolean state defining whether the heater is on or off at this moment.

Also some constant values are also required as follow:

- ***MAX_TEMPERATURE***: maximum temperature that the satellite can endure.
- ***MIN_TEMPERATURE***: minimum temperature that the satellite can endure.
- ***AVG_TEMPERATURE***: average temperature that the satellite should have.

The algorithm for this task is the following:

- Check the actual temperature value.
- If the value is lower than the ***AVG_TEMPERATURE***, set the heater on.
- If the value is higher than the ***AVG_TEMPERATURE***, set the heater off.

7 Description of the tasks: Prepare to send a command to the Arduino system.

The task is designed for preparing to send any selected command to the Arduino system,.
The commands are stated as an enum type that includes:

- ***NO_CMD***: No command has been received.
- ***SET_HEAT_CMD***: This command is used to setup the heater.
  - A boolean (***set_heater***) is needed to set the heater (on/off).
- ***READ_SUN_CMD***: This command is used to read the sun light sensor.
- ***READ_TEMP_CMD***: This command is used to read the temperature.
- ***READ_POS_CMD***: This command is used to read the position.

There is one global variable that stores the ready-to-be-send command:

- ***next_cmd_msg***: This is an ***struct cmd_msg*** variable that contains the ready-to-be-send command.

The algorithm for this task is the following:

- First, received a parameter that selects the command to be prepared.
- Then fill the variable ***next_cmd_msg*** with the corresponding information from the state of the subsystem.

8 Description of the tasks: Store a received response from the Arduino system.

The task is designed for storing the received responses by the Arduino system.
Each command has a corresponding message response, as following:

- ***NO_CMD***: This message has no response.
- ***SET_HEAT_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean)
- ***READ_SUN_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean)
    - ***data.sunlight_on***: returns the value of the sunlight sensor.
- ***READ_TEMP_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean).
    - ***data.temperature***: returns the value of the temperature.
- ***READ_POS_CMD***: The response has the following parameters:
    - ***status***: informs if the command executes correctly (Boolean)
    - ***data.position***: returns the value of the position.

There is one global variable that stores the last received response:

- ***last_res_msg***: This is an ***struct res_msg*** variable that contains the last received response message.

The algorithm for this task is the following:

- First, check out the last received response.
- Then the state of the subsystem must be updated.
- Finally, set the last response message to NO_CMD to clean it up.

9 Documentation to be submitted.

Use the web page of the course to submit the project. It is located on “Aula global”.

The files to be delivered are the following:

memory.txt | memory.pdf | memory.doc
Project memory.
autores.txt
Names and NIAs of all the students involved in the project.
arduino_code.c
Source file of the tasks for the Arduino subsystem.
arduino_code.h
Interface file of the tasks for the Arduino subsystem.
test_arduino_code.cpp
Source file of the Google unit tests for the Arduino subsystem.
i386_code.c
Source file of the tasks for the **i386** subsystem.
i386_code.h
Interface file of the tasks for the **i386** subsystem.
test_i386_code.cpp
Source file of the Google unit tests for the **i386** subsystem.

Project Deadline

Friday October 2 2023.