Middle East Technical University
Department of Computer Engineering

CENG 336

Spring 2018

Take Home Exam 4

Regulations

Due: Sunday, June 2nd 2019, 23:55.

Submission: via ODTUCLASS (NO LATE SUBMISSION)

The purpose of this assignment is to familiarize you with basic features of PICos18, a small but capable real-time operating system. You will develop several different, concurrently executed tasks with support for preemption.

Any clarifications and revisions to the assignment will be posted to the odtuclass.metu.edu.tr.

Hand Out Instructions

- `simulator.zip` : cengRoboSim simulator source files.
- `sample.hex` : A sample .hex file to test the connection between simulator and PC.
- `switchConfiguration.jpeg` : A snapshot of the board with the desired switch configurations of this assignment

Your Mission

Your mission is to write a PIC program to explore and solve a maze in a simulated environment by controlling a robot in this environment. The robot is capable of three basic motions:

- move forward
- turn right
- turn left

The robot has an 'encoder' to measure the amount of movement and rotation, and also can sense its environment via 4 sensors looking at its front, right, left and rear. You will be communicating with the simulator through serial connection, and sending/receiving different types of messages to control the motion of the robot and explore the maze via encoder and sensor information. During your exploration you will create a map of the environment. You will show this map on the LCD Module and as well as send it to the simulator.

Simulator

The robot simulator is written in Python and requires Python 2.7, `pygame` and `pyserial` to be installed on your system. The simulator is provided to you in the simulator.zip file.

For Linux, to install `pygame` and `pyserial` you can use the following command:

```
sudo apt-get install python-pygame python-serial
```

For Windows, please refer to following links for installation files:

http://www.pygame.org/download.shtml (.msi installer is available.)

https://pypi.python.org/pypi/pyserial (You should download and extract the tar.gz file and run its `setup.py` with `install` argument.)

After installing the tools or on Inek machines you can run the simulator by typing the following command:

```
python cengRoboSim.py
```

Before running the simulator, make sure that you have checked your device file name for the serial port is the same with the `DEFAULT_PORT`, which is `/dev/ttyUSB0` currently, in `cengRoboSim.py`.

If it says 'permission denied', then allow user access to device file with chmod command.

For Windows, install Python 2.7, in the installation options check pip to install the package manager. Then set up your path variable to be able to use python and pip commands in command line. Then,

```
pip install pygame pyserial
```

Don't forget to set the default port as `DEFAULT_PORT = 'COM3'` or whichever portname Windows assigns to the board.

A screenshot of the simulator is given in Figure 1. The simulator includes two regions: the Arena and the Current Map, and also displays some text about the current status. The arena covers an 800x200 pixel region and is divided into 64 50x50 pixel cells. This makes the arena a 4x16 cell region. Coordinates of top left and bottom right corners of the arena are (0, 0) and (800, 200), respectively. The robot has a 20 pixel diameter and its initial position is always (25, 25). In other words, starting position is the center of top most left cell. To construct a maze 50x50 pixel obstacle blocks (shown as black in Figure 1) are used. As a construction rule, a block can only be placed into one of 62 cells. Other construction rules are:

- A block will never be placed in the top leftmost and top rightmost cells. Top rightmost cell will be used as the target cell, whose function will be explained later in the Bonus Section.
- There will always be exactly one path from top left cell (start location) to the top right cell (target location).
- The maze will never include a loop.

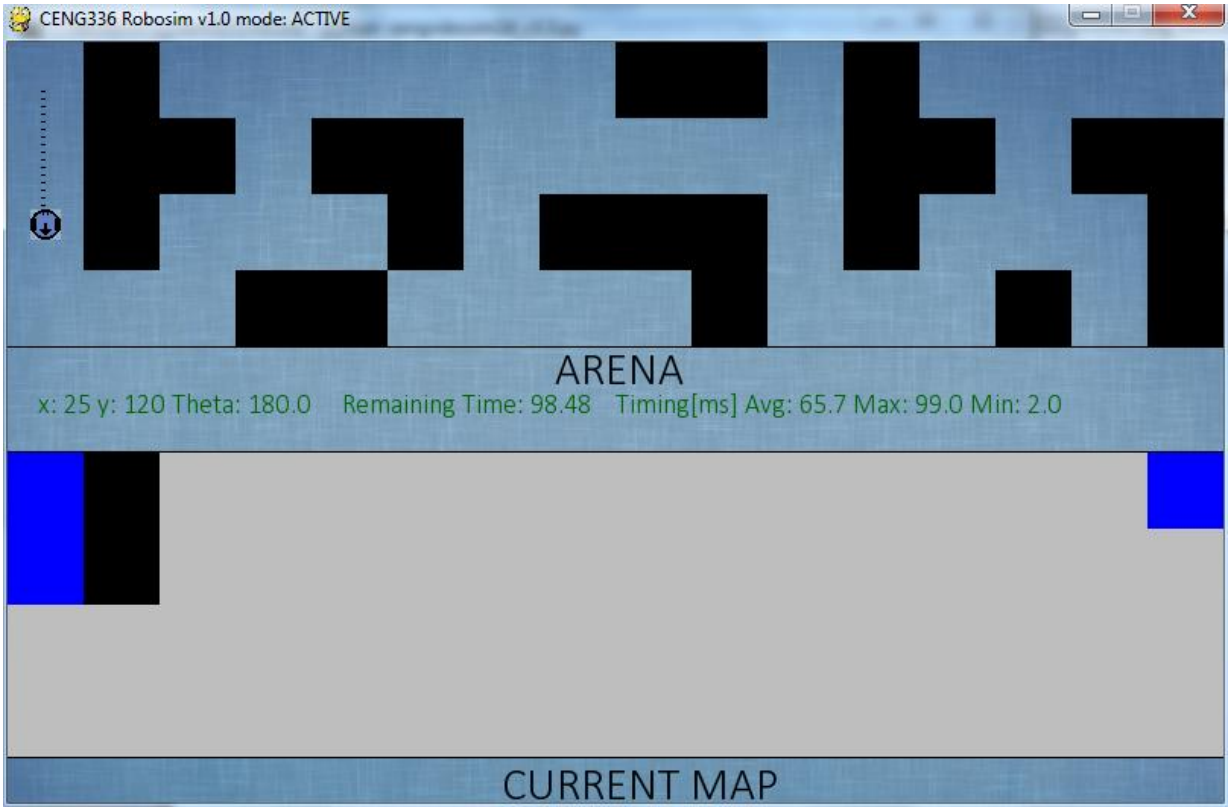

Figure 1: A screenshot of the simulator.

The initial orientation of the robot will be fixed and its orientation will be facing the bottom of the arena. This corresponds to $\theta = 180^\circ$, with θ increasing in counter-clockwise direction.

Operation Modes of The Simulator

The simulator has three operation modes:

- IDLE: In this mode, the simulator waits without sending commands or responding to the PIC until the user presses the g button on the computers keyboard. After the user presses 'g', the simulator will send a *GO command* over serial port to the PIC and switch to the ACTIVE mode.
- ACTIVE: In this mode, the simulator expects serial commands from the PIC for controlling the motion of the robot (*motion commands*) and updating the current map (*map command*). The simulator responds by *sensor responses* or *encoder responses* depending on the type of the previous motion command. There are no responses for map commands; the simulator directly updates its current map after receiving such commands.

  The simulator stays in this mode until it receives an *END command* or reaches the end of a predefined timeout.

  In this mode, the simulator also shows some status information such as the current coordinates and the orientation of the robot, the remaining time, and maximum, minimum

and average times between every two motion commands. All time differences between consecutive motion commands are also printed to the command line.

- END: When the simulator enters this mode, it prints the result of the last ACTIVE mode operation to the command line. The result includes:
  - Total operation time.
  - Correctness of current map. (Cells in the map are indicated by their row and column number. For example top most left and bottom most right cells are represented by [0, 0] and [3, 15], respectively.)
  - The result of the bonus area check (whether the robot is inside the target area or not).
  - Average, maximum and minimum durations between every two motion commands.

If 'r' is pressed in this mode, the simulator is reset to IDLE mode by clearing all of data from previous ACTIVE mode. This is only for testing purposes on the simulator. You are not expected to reset the program along with the simulator.

When user presses the 'ESC' button on the computer's keyboard, the simulation program terminates.

Commands

In general, each command starts and ends with the '$' and ':' characters, respectively. We will refer to '$' and ':' as the delimiter and terminator, respectively.

- ***GO Command :*** When the user presses 'g' in the IDLE mode, the simulator sends a GO command over the serial port to the PIC and switches to the ACTIVE mode. This command consists of the following 4 bytes (ASCII characters): **$GO:**
- ***END Command :*** This command should be sent from the PIC to the simulator to end the ACTIVE mode. You will use this command after finishing the exploration of the maze. After sending this command, the results will be displayed and you will not be able to send any other commands to the simulator (These results will be used in our grading). This command consists of the following 5 bytes (ASCII characters): **$END:**
- ***Motion Commands :*** In order to control the motion of the robot, you are expected to send commands from the PIC to the simulator while it is in the ACTIVE mode. These commands cause the robot to take forward movement or rotation steps or to stop. After receiving a command, the simulator performs the corresponding motion and possibly sends either a *sensor response* or an *encoder response* back to the PIC, after which the simulator waits for a new command. There are four types of motion commands:
  - **Move Forward:** This command moves the robot forward a certain number of pixels depending on its forward speed. After completing the movement, the simulator sends an encoder response back to the PIC. The default forward speed is 5 pixels/command. However, different integer speeds will be used during grading. This command consists of the following 3 bytes (ASCII characters): **$F:**

- **Turn Right:** This command rotates the robot a certain number of degrees in the clockwise direction depending on the configured speed (fixed and unknown). After completing the rotation, the simulator sends an encoder response back to the PIC. The default rotation speed is 9 degrees/command. However, different integer speeds will be used during grading. This command consists of the following 3 bytes (ASCII characters): **$R:**
- **Turn Left:** This command rotates the robot a certain number of degrees in the counter-clockwise direction depending on the configured speed. After rotation, the simulator sends an encoder response back to the PIC. The default rotation speed is 9 degrees/command. However, different integer speeds will be used during grading. This command consists of the following 3 bytes (ASCII characters): **$L:**
- **Stop:** This command stops the robot, causing it to perform a sensor reading. After completing the sensor reading, the simulator sends a sensor response back to the PIC. This command consists of the following 3 bytes (ASCII characters): **$S:**

- ***Encoder Response :*** This transaction is a response to the MOVE FORWARD, TURN RIGHT and TURN LEFT commands and consists of the following 4 bytes: **$EN:**

  Here **$**, **E**, **:** are ASCII characters, however **N** represents the change in the encoder value after executing the corresponding motion command. Encoder value increments by 1 for every one pixel of movement or one degree of rotation. For the MOVE FORWARD command, if the robot is not able to move because of an obstacle in front of it then N becomes equal to zero. There is no restriction for TURN RIGHT and TURN LEFT commands, they always result in a non-zero rotation, namely non-zero **N**.

- ***Sensor Response :*** This transaction is a response to the STOP command and consists of the following 7 bytes: **$DXYZT:**

  Here **$**, **D**, **:** are ASCII characters, however **X**, **Y**, **Z** and **T** represent values of 4 sensors. X is front, Y is right, Z is rear and T is left sensor as depicted in Figure 2. Note that the placements of these sensor readings depend on the current orientation of the robot, which you should keep track of using encoder readings following every STOP command.

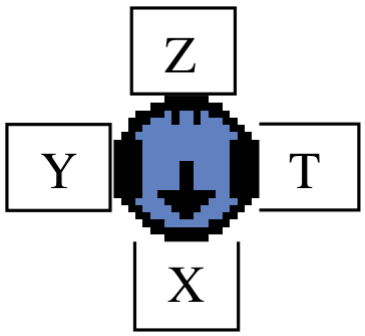

Figure 2: Sensor placements of the robot.

A sensor value can be 0 or 1, only. 0 and 1 represent no-obstacle and obstacle cases, respectively. Sensors can only sense the cells in adjacent neighborhood. They cannot sense the cells in cross neighborhood.

- ***Map Command :*** This command allows you to send perceived map information to the simulator and includes the following 6 bytes: **$MXYS:**

  Here **$**, **M**, **:** are ASCII characters, however **X**, **Y** represent cell coordinates and **S** represents information on whether an obstacle exists or not in that cell and can be 0 or 1, representing no-obstacle and obstacle cases, respectively. Any other value results in an error.

LCD Module and Button Actions

The LCD module will also show the information about the discovered blocks:

- The unexplored blocks will be represented with the "`X`" character.
- Obstacles will be represented with the "▮" character.
- Empty blocks will be represented with blanks.

The user will be able to switch between seeing the first two and the last two rows of the map by releasing the **RB4** button.

At the beginning, the LCD module should display the characters shown in Table 1, representing the unexplored first two rows of the map (the first and the last characters of the first row are blanks since it is known that they will be always empty). While your program is waiting in IDLE mode for the GO command, the user may release the **RB4** button on the board. This should refresh the LCD to display the last two rows of the map, which should be as in Table 2, all "`X`" characters.

	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Table 1: LCD displaying first two rows

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Table 2: LCD displaying last two rows

After receiving the GO command, you should start the exploration by using the commands of the simulator explained above. When you are exploring the maze, you are expected to send the map information via map commands in order to update the current map on the simulator. You should also update the LCD module. You can make use of the supplied LCD routines. While drawing the maze on the LCD module, you should write the ▮ character (0xFF) for obstacles. For the cells not including an obstacle you should write space character. An example maze is depicted below:

During program operation, if the user releases the RB4 button, the LCD will refresh to show the other two rows of the map **You must use PORTB change interrupts for this task.**

	▮						▮	▮	▮		▮	X	X	X	
	▮	▮		▮	▮						▮	X	X	X	X

Table 3: LCD displaying first two rows

	▮				▮		▮	▮	▮		▮	X	X	X	X
			▮	▮					▮		X	X	X	X	X

Table 4: LCD displaying last two rows

We recommend you use low priority interrupts for compatibility with PICos18.

You are not expected to refresh the LCD for exploring new blocks or upon RB4 release immediately to make synchronization issues more easier for you. (You may use double buffers with periodic updates on the LCD). However, the updates on the LCD should be visible before **3 seconds.**

Detailed Specifications

- **First, and most importantly, your program should make use of Real-Time Operating System principles, creating multiple tasks, setting their priorities and synchronizing them. You will be evaluated on your software design as much as the correctness of your implementation.**
- **You MUST properly comment your code, including a descriptive and informative comment at the beginning of your code explaining your design, choice of tasks and their functionality.**
- You should adjust the LCD module for **no blinking and no cursor options.**
- During exploration, you should send motion commands with a period faster than **80 ms**, with an acceptable level of accuracy $\pm$ **20 ms shifts** (Assume you decided to send commands with 70 ms intervals, the time difference between two consecutive commands should not be greater than 90ms, and should not be less than 50ms, while having an average of 70ms).
- You should send a STOP command when the robot is located precisely at the center of cells and only when the orientation of the robot is a multiple of 90 degrees. **Otherwise the simulator may not provide sensor and encoder values correctly**. You are allowed to send STOP command as much as you want. You may need such a case while you are deciding which movement you have to make next and while another task is running. But do not forget that you have a limited time to explore the maze. Note that if you want to stop for more than the frequency you had set for the commands, you have to keep sending STOP commands to the simulator.
- Timeout is set to 120 seconds, before which your program should finish exploring and solving the maze.
- Your program should be written for 40 MHz oscillator by using PICos18 operating system.
- USART settings should be 115200 bps, 8N1, no parity.

Resources

- Sample program files provided with homework.
- PIC18F8722 Datasheet
- PICos18 Documents
- PIC Development Tool User and Programming Manual
- Recitation Documents
- ODTUClass Discussions

Hand In Instructions

- You should submit your code as a single file named as the4.zip through ODTUClass. This file should include all of your source and header files.
- By using a text file, you should write ID, name and surname of **both group members and group number**.
- **Only one of the group members should submit the code**. Please pay attention to this, since if both members make submission, speed of grading process will be negatively affected.

Grading (Tentative)

Total of the homework except bonus part is 100 points. For grading we will compile and load your program to the development board. Your program will be considered for grading even if it is incomplete. We advise you to implement your program with the following goals. The list below is **not a definitive grading scheme**, but is a helpful guide about the features you should prioritize in the order they are listed.

1. **Comments and Structure (10 pts):** Comment your code, regardless of whether it works or not. Include brief but descriptive comments in your code, including a larger comment in the beginning of your file describing the structure of your program with task descriptions and their relations to one another. Your grade will greatly depend on the quality of your design, not just its functional correctness. We will evaluate your code with a white-box scheme, commenting your code will help the grader to understand your code better.
2. **Serial Communication (30 pts):**
    - Can the program properly receive GO command and send an END command? (10 pts).
    - Can the program properly send/receive motion/sensor messages? (10 pts).
    - Are the commands properly timed? (10 pts).
3. **Robot Motion (40 pts):**
    - Does the robot move proper1y? Does it sense its surroundings in the middle of the squares? (10 pts).

- Does the robot map its surroundings correctly? (10 pts).
- Is the robot able to map the maze entirely? (20 pts).

4. **LCD Display and Button Action (20 pts):**
   - Are the first two rows displayed/updated with sufficient frequency? (10 pts).
   - Is the button action implemented? (10 pts).

Note that you should structure your program modularly with different tasks, appropriate priorities and proper use of synchronization primitives. It is unacceptable to implement this system as a single task in the form of a cyclic executive with interrupts.

Bonus Part

After exploring the maze if you can manage to move the robot inside target cell then you will get an extra 15 point bonus. The robot will be considered inside the target cell if center point of the robot is at any pixel inside target cell.

Hints

- **CRUCIAL:** Since PICos18 is clearing PIE1, PIE2, RCON, IPR1 and IPR2 registers inside Kernel/kernel.asm file (instructions between lines 265-269) which are executed after your init function inside main.c, your changes on these registers are becoming ineffective. Therefore you have to configure these registers once inside a related task. If you are experiencing an unexpected stop while receiving characters from serial port (due to overrun error making OERR bit 1), this is probably caused from above issue. In that case you have to configure the bits related with receive interrupt (for example a setting like PIE1bits.RCIE = 1;) inside a task.
- In order to check the serial communication between simulator and PIC you can use `sample.hex` file. **This is not an example solution and its commands are not accurate**. Its purpose is only to check the serial communication.
- You can create your own maze and try to solve it. To do this you can edit the array holding the obstacle information named as **blocks** in `cengRoboSim.py`.
- Microchips XC series compilers are not compatible with PICos18. For inek machines, MCC18 compiler is already added, so there is nothing you have to do. For your own Linux or Windows machines, you can follow the instructions in 'PicosSettingsInMplabxManual.pdf'.

Cheating

We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations.

Cheating Policy: Students/Groups may discuss the concepts among themselves or with the instructor or the assistants. However, when it comes to doing the actual work, it must be done by the student/group alone. As soon as you start to write your solution or type it, you should work alone. In other words, if you are copying text directly from someone else - whether copying files or

typing from someone else's notes or typing while they dictate - then you are cheating (committing plagiarism, to be more exact). This is true regardless of whether the source is a classmate, a former student, a website, a program listing found in the trash, or whatever. Furthermore, plagiarism even on a small part of the program is cheating. Also, starting out with code that you did not write, and modifying it to look like your own is cheating. Aiding someone else's cheating also constitutes cheating. Leaving your program in plain sight or leaving a computer without logging out, thereby leaving your programs open to copying, may constitute cheating depending upon the circumstances. Consequently, you should always take care to prevent others from copying your programs, as it certainly leaves you open to accusations of cheating. We have automated tools to determine cheating. Both parties involved in cheating will be subject to disciplinary action. [Adapted from http://www.seas.upenn.edu/ cis330/main.html]