Middle East Technical University — Department of Computer Engineering

CENG 336

Introduction to Embedded Systems Development
Spring 2020
THE4

Due date: 08.07.2021, Thursday, 23:55

1 Objectives

The purpose of this assignment is to familiarize you with basic features of PICos18, a small but capable real-time operating system. You will learn about how to develop an embedded application with several different, concurrently executed tasks with support for preemption. You will also practice implementing asynchronous serial communication on the PIC18F4620.

Clarifications and revisions on the content, scope and requirements of the assignment will be posted to the course page discussion forum in ODTUClass.

2 Hand Out Instructions

- `simulator.zip` : cengRoboSim simulator source files.
- `sample.hex` : A sample .hex file to test the connection between simulator and PC.
- `lcdSample.zip` : Sample LCD routines and an example program to show its usage.
- `serial_echo.zip` : Serial communication example on picsim.
- `environment.yml` : conda environment listing for the robot simulator.

3 Gold Rush

3.1 Your Mission

Your mission is to write a PIC program to control a robot to collect valuable gold and silver pieces randomly appearing in a simulated environment. The robot is capable of three basic motions:

- move forward
- turn right
- turn left

The robot can sense its environment using a sensor that provides location of the metals. You will be communicating with the simulator through serial connection, sending and receiving different types of messages to control the motion of the robot, collecting precious metals and receiving sensor information. During the simulation, you will be required to use the LCD display to show the map of the environment and where valuable pieces appear.

3.2 The Game

There are two types of metals in the game: gold and silver. Initially, the area contains only the robot and some obstacles as explained below. Metals appear in random cells and at random times during the game, and they disappear after a certain amount of time. You should collect these valuable pieces before they disappear.

Gold is more valuable than silver. When you collect a gold piece you get 2 points and when you collect a silver piece, you get 1 point. Moreover, gold pieces stay on the board for a shorter time interval than silver pieces. Therefore, if a gold piece and a silver piece appear on the map at the same time, it might be better to target the gold before it disappears in order to maximize your score.

3.3 The Simulator

The robot simulator is written in Python and requires Python 2.7, with `pygame` and `pyserial` packages. If you are not sure how to install them, here is a way using Anaconda:

1. Download Anaconda using this link.
2. Install Anaconda, with the GUI on Windows and the shell script for Linux. For Windows, make sure you add it to PATH by checking the related checkbox in the GUI. For Linux, you can add it to PATH by typing yes to the prompt “Do you wish the installer to initialize Anaconda3 by running conda init?”.
3. Change your current directory to the directory where `environment.yml` is located in the command line (with `cmd` in Windows and your default shell in Linux) and then run `conda env create -f environment.yml`. This should create an environment named `ceng336`.
4. Each time you’d like to use this environment you will need to activate it with `conda activate ceng336`. Your command line prompt will change upon this action. After activation, simply type `python cengRoboSim.py` to start the simulator (you will first need to set your port name in the Python script as described below).

Before running the simulator, make sure that you have checked your device file name for the virtual serial port in PICSimLab is the one paired with the `DEFAULT_PORT`, which is `/dev/tnt3` currently (paired with `/dev/tnt2` in Linux), in `cengRoboSim.py`. Here are example virtual port pairings described in the PICSimLab manual for Linux and Windows. For Windows, you will also need to enable buffer overruns through the Setup GUI, which can be found from Start Menu > com0com. After setting the port of the robot simulator, you will need to select the other part of the port pair from PICSimLab, from File > Configure.

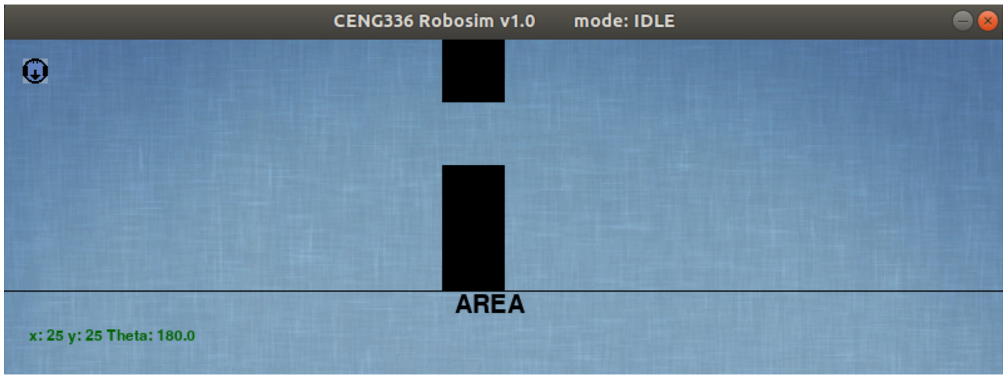


Figure 1: A screenshot from simulator.

A screenshot of the simulator is given in Figure 1. The area covers an 800x200 pixel region and is divided into 50x50 pixel, 64 cells. Thus, the region is composed of 16x4 cells. *Pixel coordinates* of top left and bottom right corners of the region are (0, 0) and (800, 200), respectively. *Cell coordinates* of the top left and the bottom right 50x50 cells are (0, 0) and (15, 3), respectively. The robot has a 20 pixel diameter and its initial position is always (25, 25) in pixel coordinates. Its orientation will be facing the bottom of the area.

The map area contains fixed 50x50 pixel obstacles situated at the center of some cells. (shown as black cells in Figure 1) that will not change throughout this assignment (in grading also). Other construction rules are:

- Metals appear at the center of the cells. Metals do not appear on top of the obstacles.
- There will always be at least one path from the current position to the metals.

Operation Modes of The Simulator

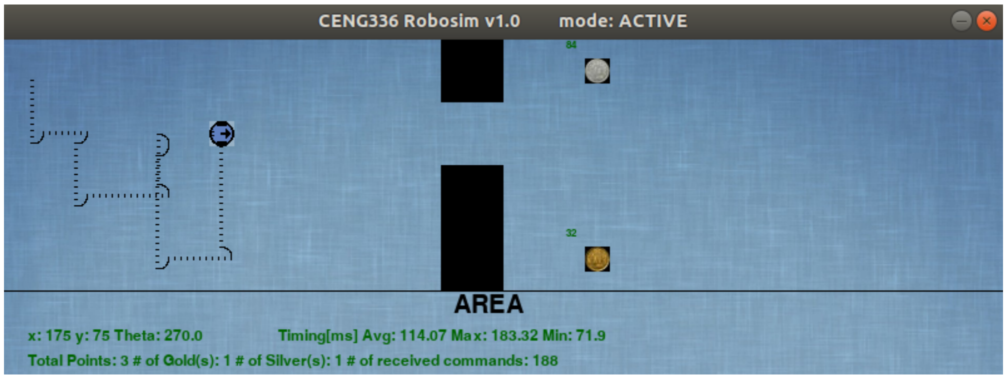
The simulator has three operation modes:

- 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 response* 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*). In addition, there will be a command to pick a metal and a command to end the simulation.

One other aspect of this mode is that the *robot simulator time* is incremented upon receiving a command. As the robot simulator time ticks away, the lifetime of the coins decrease and after a certain number of commands the simulator simply times out. The lifetime of the coins can be observed with the numbers at the top left of the coins that appear.

The simulator stays in this mode until it receives an *END command* or reaches the end of a predefined timeout (in terms of the number of received commands, i.e. robot simulator time). This timeout is guaranteed to be later than the time the last coin disappears. **You should strive to terminate the simulation using the *END command*.**

In this mode, the simulator also shows some status information such as the current coordinates and the orientation of the robot, total points, number of picked silver coins, number of picked gold coins, maximum, minimum, average times between every two command, and the number of the received commands. All time differences between consecutive commands are also printed to the `stdout`.



```
x: 175 y: 75 Theta: 270.0          Timing[ms] Avg: 114.07 Max: 183.32 Min: 71.9
Total Points: 3 # of Gold(s): 1 # of Silver(s): 1 # of received commands: 188
```

Figure 2: Simulator in ACTIVE mode.

- END: When the simulator enters this mode, it prints the result of the last ACTIVE mode operation to the command line. The result includes:
  - The number of picked gold pieces, the number of picked silver pieces and the total points you earned.
  - Average, maximum and minimum durations between every two motion commands.

  25 coins will appear throughout one game.

Commands (Issued by the PIC)

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

- **END Command:** This command should be sent from the PIC to the simulator to terminate the ACTIVE mode. You will use this command after all metals (25 pieces) are displayed and disappeared. After sending this command, the results will be displayed and you will not be able to send any other commands to the simulator. 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. There are four types of motion commands:
  - **Move Forward:** This command moves the robot 5 pixels forward. This command consists of the following 3 bytes (ASCII characters): **$F:**
  - **Turn Right:** This command rotates the robot 9 degrees in the clockwise direction. This command consists of the following 3 bytes (ASCII characters): **$R:**
  - **Turn Left:** This command rotates the robot 9 degrees in the counter-clockwise direction. This command consists of the following 3 bytes (ASCII characters): **$L:**
  - **Stop:** This command does nothing. You can use it to simply advance the simulation. This command consists of the following 3 bytes (ASCII characters): **$S:**

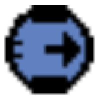


Figure 3: Appearance of the robot when its orientation is 0 degrees (positive counter-clockwise).

- **Pick Command:** This command allows you to pick any coin if the location of the robot coincides with the metal in terms of pixel coordinates. This command consists of the following 3 bytes (ASCII characters): **$P:**

Responses (Issued by the Robot Simulator)

- **GO Response:** When the user presses 'g' in the IDLE mode, the simulator sends a GO response over the serial port to the PIC and switches to the ACTIVE mode. This response consists of the following 4 bytes (ASCII characters): **$GO:**
- **Sensor Response:** At some point in robot simulation time, the simulator may generate a metal. If this is the case, the information about this metal is sent as a reply to an arbitrary command of the PIC. The robot simulator sends the following 6 bytes: **A\underline{\textbf{XYT}}$:**

  Here **$**, **A**, **:** are ASCII characters, however the bytes $\underline{\textbf{X}}$, $\underline{\textbf{Y}}$ represent cell coordinates of the metal (as their numeric value — not their ASCII value). and the byte $\underline{\textbf{T}}$ represents the type of the metal: if it is gold it should be 1, if it is silver it should be 0 (as their numeric value — not their ASCII value). When this command is received in PIC, you should show the metal with the given coordinates on the LCD module.

LCD Module and Button Actions

At the beginning, the LCD Module should show the following screen and your program should wait for a GO command to start the game.

						G	O	L	D						
						R	U	S	H						

Table 1: Initial LCD screen

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.

When a **sensor response** is received, you should update the LCD module accordingly. When you pick the metals, you are expected to simultaneously update the LCD module. Also, when the time expires for a metal, it should not be shown on the LCD module anymore. You can make use of the supplied LCD routines. While drawing the area on the LCD module, you should write the █ character (0xFF) for obstacles, the "**G**" character for gold metal and the "**S**" character for silver metal. For the cells that not include an obstacle or metal, you should write a space character on that cell. You are not required to show the robot on the LCD module. An example area is depicted in Tables 2 and 3.

							█								
														S	

Table 2: LCD displaying first two rows

				G			█								
							█								

Table 3: LCD displaying last two rows

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.**

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

4 Detailed Specifications and Submission

- **Firstly, and most importantly, your program should make use of Real-Time Operating System principles, creating multiple tasks, setting their priorities**

and synchronizing them using RTOS primitives such as semaphores. You will be evaluated on your software design as much as the correctness of your implementation.

- You must have at least 2 non-trivial tasks that run concurrently in your application. One such task could be the one refreshing the LCD screen.
- **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.** %5 of your grade will depend on the quality of your comments.
- You should adjust the LCD module for **no blinking and no cursor options**.
- During ACTIVE mode, you should send every command with a fixed period of your choice between 50-150 ms. This period may fluctuate in the robot simulator as PIC-SimLab may not able to run real-time, however, your average rate should be only slightly above the value you set in software. This ”software” value should be fixed and clearly observable in your code and the MPLAB simulator using the Stopwatch feature.
- 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 number of commands to pick any metal.
- It is recommended to move the robot with ”L shaped” moves. In other words, you are recommended to rotate the robot at multiples of 90 degrees; to move the robot when its orientation is a multiple of 90 degrees and from the center of a cell to the center of another. Otherwise, you can have difficulties to align the robot at center of the cells. If you prefer to move otherwise, this may require complex and probably time consuming trigonometric and floating point operations. Nevertheless, you are free to decide on the movement style.
- You should pick at least **3 gold** and **3 silver** metals (which are guaranteed to appear). Otherwise, you will be considered as unsuccessful. Moreover, your total point will be checked at the end of the game. The total point is calculated as follows:

  ```
  (number of the picked gold pieces x 2) + (number of the silver pieces)
  ```

- For each metal there is a timeout in terms of number of commands you sent. Gold metals disappear when you send 90 commands. In other words, you should send **pick command** before any other 90 commands (You can send the **pick command** as the 90th command. If the last command is not **pick**, then the gold metal disappears). The timeout for silver metal is 100 commands. For example, you can pick the silver metal on the 95th command, but you cannot pick it on the 101st.

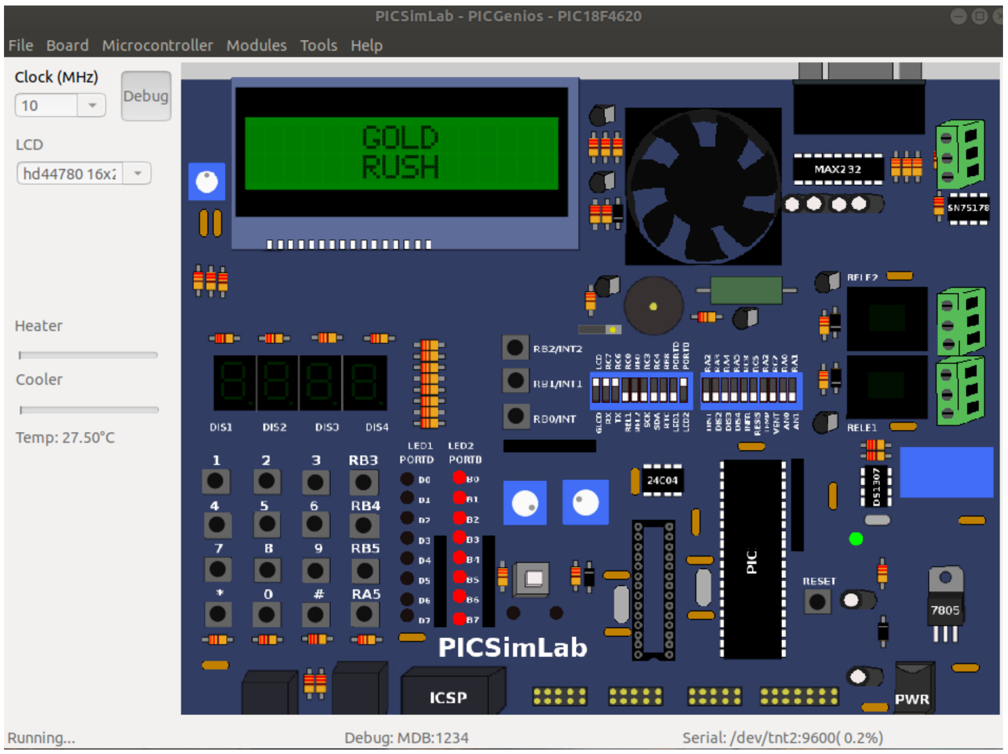
- Your program should be written for **10 MHz** oscillator by using PICos18 operating system, for PIC18F4620.
- USART settings should be **asynchronous, 9600 bps, 8N1, no parity**.
- You should use interrupts for serial communication.
- You should submit all of your headers and source files in a zip file with the name `eXXXXXXX_the42021.zip`.

5 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.
- `picsim` does not seem to comply with the datasheet in terms of behavior in asynchronous serial transmission interrupts. Normally, after you set the `TXSTAbits.TXEN` bit for the first time, PIC goes into interrupt immediately as the shift register transmitting the bits is empty (without needing to load TXREG). However, in PICSimLab you need to explicitly load the `TXREG` and then set `TXSTAbits.TXEN` for the very first character you want to send in order to trigger an interrupt. Another difference is that you need to unset `PIR1bits.TXIF` just like your other interrupt flags in the ISR for picsim. Normally, this flag is not clearable in software according to the datasheet (you either load another char into `TXREG` or unset `TXSTAbits.TXEN` to clear the interrupt). `serial_echo` project is attached as an example for these phenomena.
- Make sure the dip switches for connecting LCD to PORTE/D and RX/TX pins of the serial port to RC6/7 are on (Figure 4).
- In order to check the serial communication between simulator and PIC you can use `sample.hex` file. **This is not an example solution.** Its purpose is only to check the serial communication. This program moves the robot on the leftmost column down and then ends the simulation.

6 Cheating

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


PICSimLab - PICGenios - PIC18F4620

Clock (MHz) 10

LCD hd44780 16x2

Heater

Cooler

Temp: 27.50°C

Running...

Debug: MDB:1234

Serial: /dev/tnt2:9600(0.2%)

Figure 4: Board Settings for the assignment.

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 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 or code directly from someone else - whether copying files or typing from someone elses 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 elses 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