Lab 2

Line-following Robot in simulator using Ada

Behnam Khodabadeloo

20 Nov. 2024

UPPSALA UNIVERSITET

Lab Instruction

- Find lab instructions on the assignment page
- Lab sessions: November 22nd, 13:15-17:00, 6K1123 & November 25th, 8:15-12:00, 6K1123
- Lab sessions not mandatory, but recommended (TAs are there)
- Lab 2 has a **mandatory demo session** on December 2st, 13:15 - 17:00, 6K1123
- Lab 2 **deadline**: December 6th, 23:59

UPPSALA UNIVERSITET

Lab Goals

- Part 1: To **introduce the lab environment** and ensure that everything is working in a simple setup.
- Part 2: Implement **a fully controllable car** (using the directional keys on the keyboard). Use the principles of event-driven scheduling.
- Part 3: Implement **a line follower robot with obstacle detection**. Use event-driven scheduling (described in part 2) or periodic scheduling (described in part 3) to implement this part of the lab.

UPPSALA UNIVERSITET

Lab Submission

- **Demonstration**
- A group member must **upload the code** before the deadline (no report needed for Lab 2)
- Late submission budget: 8 days
- A late submission subtracts corresponding days from all group member's budget
- No late submission possible without enough budget!
- Special cases: contact the TA for extension

- Discussions with other groups are ok, but don't copy
- Don't cheat!

UPPSALA UNIVERSITET

Lab Grading

- Grading based on **Demonstration + Code Submission**
- **P**ass/**F**ail
- (On Canvas) Grades:
  - "Completed",
  - "Incomplete, under supplementation"
- "Incomplete" labs must be supplemented. To pass, you must revise the reports or codes based on our comments and submit again.
- Grade is intended for the whole group, but individual F is possible if any member is clearly not contributing

UPPSALA UNIVERSITET

Lab Environment

- Simplest setup: virtual machine + our VM image. A virtual machine image with all code skeleton provided
- Ångström computer rooms provide Windows machines.You can download virtualbox from software center. Remember to save your file when using lab machines
- For Mac (Apple Silicon) users: try building a ubuntu docker image. Remember to commit your change to the image or use the same container every time. (As far as I know: no GUI; works for lab 1 and 3)

UPPSALA UNIVERSITET

Lab Environment

- Required Tools: **Webots & GNAT**
- Assuming GNAT is installed (as part of Lab 1)

- The provided rebot environment was built for Webots 2020b (best to download this version).
- Download and install Webots from link provided in the lab instructions page.

UPPSALA UNIVERSITET

Files

- Skeleton Files: contains two folders named **e-puck** and **skeleton**.
- The file e-puck/worlds/**e-puck line.wbt** is used to open the simulation world on Webots.
- Five files in the skeleton folder
  - main.adb: The entry point of the program
  - tasks.ads: The specifications of your tasks
  - tasks.adb: The body (or implementation) of your tasks
  - webots_api.ads: The specification of API functions and procedures for accessing Webots input/output data
  - webots_api.adb: The implementation of the Webots API which communicates with Webots using TCP/IP ports

UPPSALA UNIVERSITET

Part 1: Hello Webots!

Goal:

- Set one or both motor speeds to some value within the range [-999,999] (negative values change direction), so that your robot has some visible movement in the arena.
- Read one of the three ground-facing light sensor values and display it repeatedly with a delay of 100ms in between.
- Compile your code

UPPSALA UNIVERSITET

Part 1: Hello Webots!

Demo

UPPSALA UNIVERSITET

Part 1: Hello Webots!

Clarifications

In order to ensure smooth operation while testing your programs (i.e., not to wait for the TCP/IP port to timeout), pause Webots simulation before stopping the Ada program. The Ada program can then be stopped by pressing ctrl+c in the terminal.
To run your program again, first reset the Webots simulation either by pressing the "Reset Simulation" button or by using its shortcut ctrl+shift+T in Webots. After the simulation is started you can run your Ada program again.

UPPSALA UNIVERSITET

Part 2: Event-driven scheduling

Goal:

The Webots robot that can be **manually controlled**

- using the keyboard directional keys (as long as at least one of the keys is pressed)
- and while it is not over a black line (in which case it should immediately stop).

UPPSALA UNIVERSITET

Part 2: Event-driven scheduling

Create a **protected objected** named **Event**

– A single entry and an enumeration of possible EventIDs

– Used by different tasks to communicate between them.

```
type EventID is (UpPressed, DownPressed, ...) -- events

protected Event is
       entry Wait(id : out EventID);
       procedure Signal(id : in EventID);
private
        -- assign priority that is ceiling of the
        -- user tasks' priorities
       current_id : EventID;     -- event data
       signalled : Boolean := false;  -- flag for event
end Event;

protected body Event is
      entry Wait(id : out EventID) when Signalled is
      begin
         id := current_id;
         signalled := false;
      end Wait;

      procedure Signal(id : in EventID) is
      begin
         current_id := id;
         signalled := true;
      end Signal;
end Event;
```

UPPSALA UNIVERSITET

Part 2: Event-driven scheduling

- Implement a task **EventDispatcherTask** that call the Webots API function to read the button presses. If the state has changed, it should release the corresponding event by using signal procedure of the Event protected object.
- Implement a task **MotorControlTask** that waits for an event in an infinite loop and update robot motor speeds based on the received event.
- Assign a **lower priority** to EventDispatcherTask over MotorControlTask. Otherwise, the infinite loop containing the sensor reading would might make the system completely busy
- **Extend the program** to also react to one of the three ground-facing light sensors. The car should stop when the light sensor detects that the car is on top of a black line.

UPPSALA UNIVERSITET

Part 2: Event-driven scheduling

Clarifications

- The job of `EventDispatcherTask` is just to create the events signaling key presses/releases and line detection, *independently* from each other.
- All logic should happen in `MotorControlTask`, i.e., when to move or not, depending on the current state. This task must not read the sensors by itself and must not communicate with `EventDispatcherTask` by any other means except through the `Event` protected object, i.e., shared variables etc. are not allowed.
- A frequent mistake is that the periodic `EventDispatcherTask` generates an event each time it is executed, reporting the current state of the sensors. Instead, it should only generate events at *state changes*: one (and only one) in the moment when a button is pressed down and one when it is released[a]. The same is true for the light sensor: when the black line is detected, generate *one* event. Doing otherwise might spam the event system with redundant information.

UPPSALA UNIVERSITET

Part 3: Line Tracker

Goal:

A robot that can simultaneously:

- follow the black line using the light sensors.
- keep a constant distance to detected objects in front of it using the distance sensors

Note: Your implementation should either use periodic scheduling or event-driven scheduling

UPPSALA UNIVERSITET

Part 3: Periodic Scheduling

- High-level structure: We have several tasks that are scheduled periodically, with different periods
  – A task LineFollowingTask that reads the light sensors
  and sends commands to MotorControlTask.
  – A task MotorControlTask that only takes care of setting the speeds of the motors.
  – A task DistanceTask that reads the distance sensor and sends commands to MotorControlTask.
  – A task DisplayTask that displays some status information

UPPSALA UNIVERSITET

Part 3: Periodic Scheduling

Communication between the tasks:
– Use global data structures which can be modified by some tasks, and read by others

```
protected SharedData is
   procedure Set (V : Integer);
   function Get return Integer;
private
   --  Data goes here
   some_data : Integer := 0;
end Obj;

protected body SharedData is
   --  procedures can modify the data
   procedure Set (new_data : Integer) is
   begin

      some_data := new_data;
    end Set;

    --  functions cannot modify the data
    function Get return Integer is
    begin
       return some_data;
    end Get;
end SharedData;
```

UPPSALA UNIVERSITET

Part 3: Periodic Scheduling

Clarifications

- You may use all three available light sensors.
- The obstacles will be the solid boxes already present in the arena. During demonstration, you will be asked to manually place these boxes (and possibly move them) in front of your robot. Alternatively, we may also provide a script to do this automatically.
- The robot does not have to drive backwards. The obstacles it has to follow will not move backwards, and if your robot gets too close, it is sufficient to just slow down/stop.
- Your robot does not need to increase its normal speed to catch up to an obstacle if it is moving too fast; however, if the obstacle is moving at a slower constant speed, your robot should also keep moving behind it while keeping a constant visible distance[a] behind the obstacle.
- Don't assume a direction on the track; however, your robot will be placed alongside a black line. Your robot must work both clock- and counterclockwise equally well.

UPPSALA
UNIVERSITET

The end
Good luck