Software Testing 2019

Assignment SWT2. System testing

Deadline: Sunday 19 May

Aim. The aim of this assignment is to experience the testing of software which is embedded in physical systems.

Method. You will work in groups of 3-4 students. Your task is to code and test a MATLAB software controller for a Lego Mindstorms ev3 based vehicle.

You must hand in your report (pdf) in Canvas until **Sunday, 19 May, 23:59.**

Don't forget to write your names on the report and the time you spent on it.

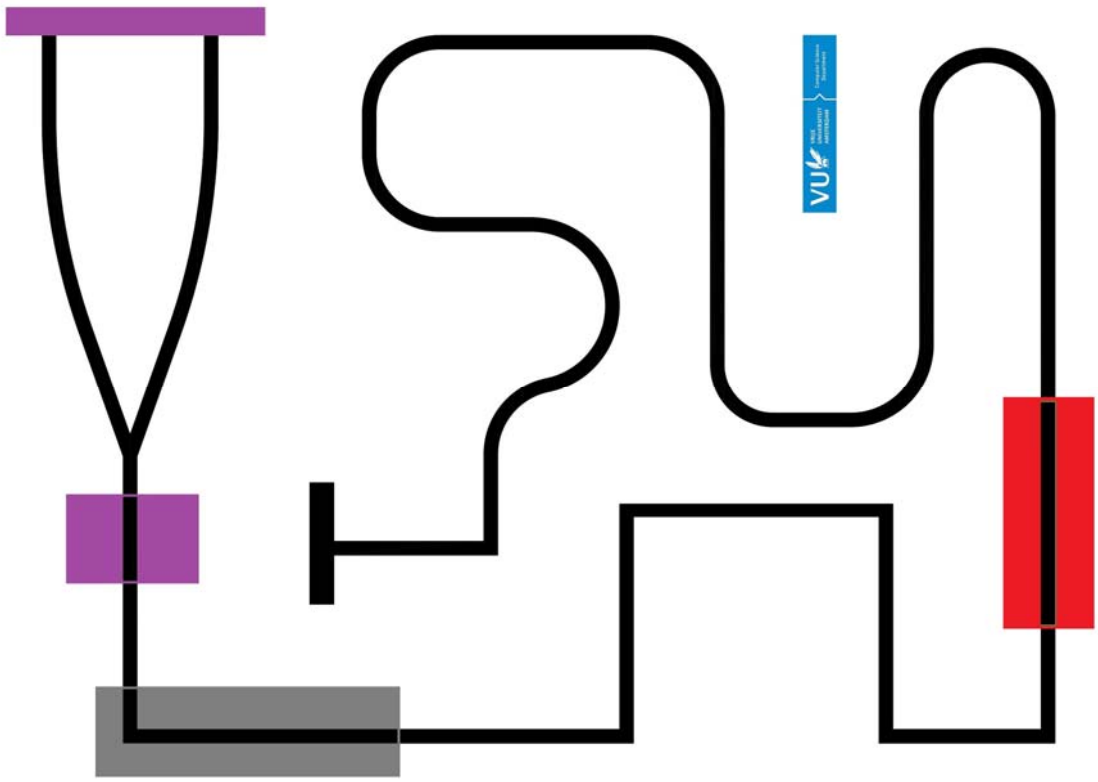
Your assignment

The system-under-test (SUT) is a simple cyber-physical system, consisting of a Lego Minstorms ev3 robotic vehicle, controlled by a MATLAB software agent. Your task is to code the software agent and test the system as a whole. The system has to satisfy the following user requirements (see figure below):

```
The robot shall follow a black line on a one-way track.
The robot starts at the red T-junction.
The track has three zones, where the following rules apply:
   1. In a white zone, the robot drives with NORMAL speed.
   2. In a gray zone (rainy weather), the robot drives with a LOW
      speed.
   3. In a green zone (highway), the robot drives with HIGH
      speed.
At the end of the track, the robot ends its journey in a free
parking lane.
Collisions should be avoided at all times.
A journey from start to the end, with no obstacles on the way,
should not take more than five minutes.
```

1. Write a set of system requirements and divide them in two groups (functional and non-functional).
2. Get familiarized with MATLAB and the physical system by following the tutorial in the Appendix.
3. Decompose the functionality and program in MATLAB units for each small piece of functionality and save them in separate .m files. For each unit, generate unit test cases and run them.
4. Integrate the modules and write integration test cases. Specify which integration methods did you use (big bang, feature-based)? Did you think about fault injection such as simulation of a sensor failure?
5. For system testing, imagine some end-to-end thread scenarios and execute them. All tests should be traceable to system requirements.
6. Acceptance testing will be done by us, the instructors. Therefore you should show to one of us that the system is working as expected, and send us a video showing your robot in action.

You have to submit in Canvas a report, containing (1) a description of your test strategy and your test cases for integration and system testing, and (2) a short evaluation of your system testing process (What did you like? What you did not like? What was difficult? Did you have any surprises created by the fact that you tested a real physical system? Where did you spent most of your time? What did you learn?).



Practical notes. You will need a laptop (preferably Windows) with MATLAB 2017b or later installed. We have a VU campus license for MATLAB(see https://download.vu.nl/). Also, you will need the Lego Mindstorms ev3 toolbox.

Appendix.
Programming the LEGO Mindstorms EV3 vehicle

1.The vehicle

The LEGO Mindstorms EV3 robotic kit consists of an intelligent unit (or brick), together with a set of different sensors and actuators, shown in Fig. 1.

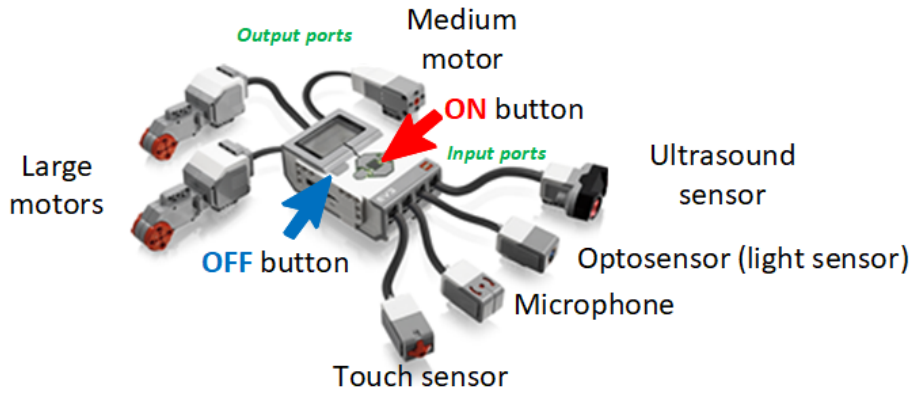


Fig. 1. Overview of the LEGO Mindstorms EV3 system.

Each sensor can be connected to any of the four **input ports**, situated at the bottom of the LEGO brick, marked with 1, 2, 3, and 4. The actuators can be servomotors or LEDs, connected to one of the four **output ports,** situated at the top of the LEGO brick, labeled with A, B, C and D.

The vehicle we will use in this tutorial (Fig. 2) has one ultrasound sensor, two light sensors and as actuators a speaker, an LCD display and two servomotors. The speaker can be used to play music.
Note: You are not allowed to change anything about the robot except these cable connections.

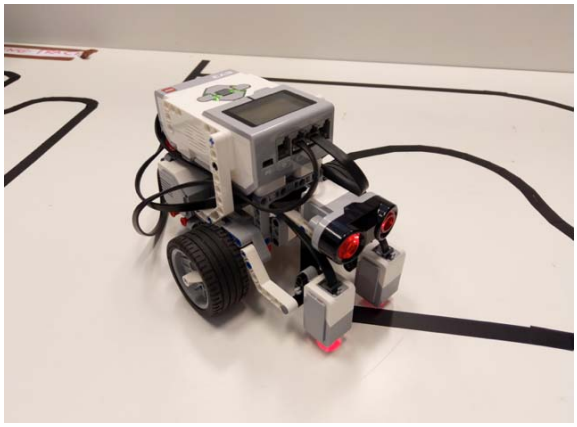


Fig. 2. The LEGO Mindstorms EV3 robot used in this lab.

2. Connecting the LEGO Mindstorms robot to your computer

Starting from 2017b, MATLAB offers support for LEGO Mindstorms ev3. All functions are described on this site:
https://nl.mathworks.com/help/supportpkg/legomindstormsev3io/referencelist.html?type=function

Note: The LEGO ev3 toolbox is installed on all computers in U130. If you are using your own laptop, you will have to install it separately, by using the MATLAB APPS-Install apps menu. Here you can find out how. https://nl.mathworks.com/hardware-support/lego-mindstorms-ev3-matlab.html

Examine your LEGO Mindstorms ev3 vehicle and try to identify the main unit (or "brick"), the input and output ports, the sensors and the actuators. Write down which sensors and actuators you see, and the ports to which they are connected.
Your MATLAB program which controls the vehicle will be running on the computer. It will receive data from the sensors, and will send back commands via an USB serial communication cable.
Before sending and receiving data, the computer has to be connected to your robot. This can be done in two steps, as follows:

Step 1. Use an USB cable to connect the robot to the PC. Turn ON the LEGO robot by pressing for a couple of seconds the dark gray square button in the middle of the front panel. If it does not start, maybe you forgot to connect the USB cable.
Note: You can switch it off by pressing the OFF button.

Step 2. Start MATLAB 2017b. Now type in the MATLAB command window the following command:

```
>>myrobot = legoev3('usb');   %Opens a new connection to a EV3
                              %robot on a USB port
```

MATLAB uses a "handle" created by the legoev3 function to communicate with the robot. A handle is just a variable containing information about the EV3 device, like for example which port it is connected to. All of the provided code is set up to use a "default handle," which is specified with the legoev3 function. You can easily verify that everything is correctly connected by - for example - "asking" the robot to play a tone. This can be done with the following command:

```
>>beep(myrobot);    % Generate a beep
```

Finally, when you're ready to disconnect the EV3, you can also learn how to properly close the connection by clearing the variable `myrobot` from the MATLAB workspace.

```
>>clear myrobot;  % Close the connection
```

TIP: if you experience any connection issues after disconnecting and reconnecting the USB cable, restarting MATLAB could solve these issues.

3. Activating the actuators

Open an USB connection to the robot, as we have already done in Section 2.

3.1. The speaker

The speaker is an actuator that can be used to play a tone. MATLAB has a function for this, called playTone (myev3,freq,duration,volume), where the frequency of the sound is expressed in Hz, from 250 to 10000Hz, the tone duration is expressed in seconds, from 0 to 30s, and the volume is a level between 0 and 100.

Try this command to play an A 440 note for 5 seconds and an average volume.

```
>> playTone (myrobot, 440, 5, 50);
```

3.2. Drive the motors

First, we will try to drive just a single motor – for example the one connected to output port A. To do this, we need to create a **motor object**. Using this object, we will be able to send various commands to the motor. To create such an object, which we'll call `mA`, just use the following command:

```
>>mA = motor(myrobot, 'A'); %create an object for the motor A
```

The object `mA` provides various properties and methods to control the EV3 motor on port A. First, you probably just want to see if the motor works at all:

```
>>mA.Speed = 50;  % we start with half the maximum power
>>start(mA); % this is actually the moment we start the motor
```

Motor A should now run indefinitely. To stop it, use the `stop()` method of the mA object:

```
>>stop(mA);
```

This will turn off the power, causing the motor to come to a soft stop (so-called "coasting"). If we wanted to brake all of a sudden at once, we could instead have used:

```
stop(mA, 1);
```

If you do this, the EV3's active brake will be enabled, which makes the motor stop immediately after receiving the command. The brake works actively to stop the motor moving; it's surprisingly strong, but it also consumes a lot of power.

A negative speed value mA=-50 will drive the motor in reverse.

If we want our robot to drive forward with both motors simultaneously, connected to output ports A and C, then we need to create two different motor objects.

```
>> mA = motor(myrobot, 'A');
>> mC = motor(myrobot, 'C');

>>mA.Speed = 50;
>>mC.Speed = 50;
>>start(mA);
>>start(mC);
```

When you want to stop both motors, you can type in the command:

```
>>stop(mA);
>>stop(mC);
```

4. Reading the sensors

Open an USB connection to the robot, as we have already done in Section 2.

As we already mentioned, our vehicle is equipped with an ultrasound sensor and two colour sensors, shown in Fig. 3. In this section, we will show you how to read data from these two sensors.

Fig. 3. The LEGO Mindstorms EV3 ultrasound (left) and colour sensor (right).

4.1. Reading an ultrasound sensor

An ultrasound sensor measures the distance to an object in front of it. The result is returned in [meters]. Let us assume that the ultrasound sensor is connected on input port 1. To start, we have to "open this port for reading" using the command:

```
>>myUltrasonicSensor = sonicSensor(myrobot);
```

We read a sample can by asking for a single measurement from the sensor, and store it in a MATLAB variable called `val`.

```
>>val = readDistance(myUltrasonicSensor);
```

We can display the measured value `val` on the screen with the MATLAB function `display`:

```
>> display (val);
```

Note that this is just one value, measured at a certain moment in time. If you need more values, then you will have to call again the function readDistance.

4.2. Reading a colour sensor

The ev3 colour sensor can be used to measure light intensity. It can function in two modes: a **passive** one, where it measures the ambient light intensity, and an **active** one, where it uses a light source (LED) to shine on an object, and thus measure the intensity of light reflected from a surface.
We open the light sensor connected for example to input port #1 by typing in this code:

```
>>myColourSensor = colorSensor(myrobot,1);
```

Ambient light can be measured by reading the sensor in Passive mode, using the command:

```
>> ambient=readLightIntensity(myColourSensor)
```

As a result, MATLAB will display the light intensity value measured by the sensor, an integer in the range from 0 to 100.

Try to create shadow around the sensor, and read again the measured light value. What do you notice?

We can also measure the reflected light by reading the sensor in Active mode, with the command :

```
>>reflected = readLightIntensity(myColourSensor,'reflected')
```

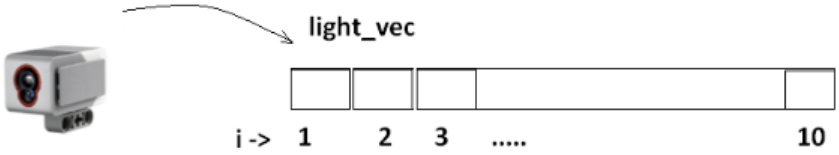
Let's collect 10 readings from the light sensor, with an interval of 0.5ms, and then display the measured values (the samples). For each reading, the code will perform the same three steps: **measure – display – wait**. Repeating the whole sequence for a number of readings can be achieved by using a MATLAB control statement called the `for` loop.
The sampling period (the time distance between two measurements) can be represented in MATLAB with the command `pause(xx)`, that waits for xx milliseconds. In our case, we need to measure 10 times and the sampling period is Ts = 0.5 ms. All this will result in the following code:

```
for i=1:10
  val = readLightIntensity(myColourSensor,'reflected');
    % read the sensor in variable val
  display (val);              % display the variable val on the screen
  pause (0.5);                % wait for 0.5 s
end
```

If you want to **interrupt a command** that MATLAB is running, type <ctrl> + <c>.

In this experiment, we measured 10 values and we displayed them, but we will lose them after the `for` loop is finished, which means we cannot process them further. We can improve the situation by repeating the measurements and also storing the measured values in a numerical vector, called for example `light_vec.` Try to test this code:

```
for i=1:10
  light_vec(i)= readLightIntensity(myColourSensor, 'reflected');
       % read sensor and store in vector element i
  pause( 0.5 );  % wait for 0.5 ms
end
```



Now the data is safely saved in the memory. You can then process further this vector however you want (for example, by plotting it). You can see that the new vector light_vec popped up in the MATLAB variable space window.

4.3. Saving the measured data in a file

In the previous experiment, we measured 10 values and stored them in a vector. However, they will still be lost as soon as we close the MATLAB session. It could be even safer to **save these values in an ASCII file**, so that they can be retrieved later any time for further processing. This command:

```
>> dlmwrite ('data.txt', light_vec) ;
```

will save the data from the `light_vector` into a file, called data.txt in ASCII format.

You can display the contents of the file in MATLAB with:

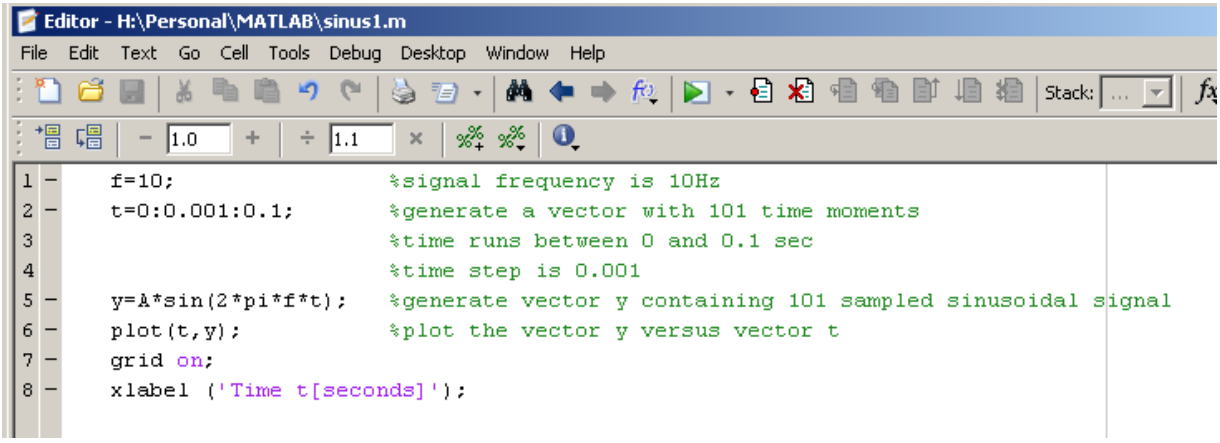
```
>> type('data.txt')
```

you can load the file content back into a vector using the MATLAB command `dlmread`.
For example this command will load the data from the file in a new MATLAB vector, called new_vec.

```
>> new_vec = dlmread ('data.txt');
```

5. Scripts

Now that you are familiarized with the robot, you should start writing **scripts** or an **M-files** instead of typing commands at the MATLAB prompt, because those commands will disappear as soon as you close the MATLAB session. To create an M-file, select New-Script and an Editor window will open. You can then type in the commands you need, and you can start by just copying and pasting them from your Command window. Below you can see an example of a MATLAB script called sinus1.m.



You can run a script by pressing the green arrow button in the Editor Menu, or by typing its name at the MATLAB prompt.

```
>> sinus1 ;
```