ePic2 v2.1 Documentation

Julien Hubert & Yannick Weibel

November 5, 2008

1 Introduction

The ePic2 software is a toolbox to interface Matlab with an e-puck (for additional information see http://www.e-puck.org). It contains a graphical interface and a set of primitives allowing to send and receive commands between the e-puck and Matlab. It will be used in the laboratories of the course Robots Mobiles (http://moodle.epfl.ch/course/view.php?id=261). The goal of the software is to provide students with the necessary tools to explore the topic of mobile robotics while focusing on high level behaviors or strategies.

The next section will present the commands to interact with the e-puck available from the Matlab command line. Following, the graphical interface and its possibilities will be described.

2 Quickstart

This section intends to give you the quick steps needed to start working with ePic2.

1. **epic=ePicKernel;** creates a class containing all the variables needed to command the e-puck
2. **epic=connect(epic,'COMXX');** will connect epic to an e-puck accessible through the serial port COMXX
3. **epic=update(epic);** updates the status of the physical e-puck and of the epic class. It transfers data from and to the e-puck.
4. **epic=disconnect(epic);** breaks the connection between the e-puck and Matlab.

3 Matlab Commands

3.1 The ePicKernel class

ePic2 is based on a class containing all the information related to the e-puck. This class contains the values of the sensors, their rate of update, the commands that should be sent to the e-puck and the inner parameters of the ePic2 interface.

To act on the e-puck, the class contains a set of functions setting some inner variable related to its control. Those will be detailed in the following sections.

The class is created by calling *epic=ePicKernel;*.

3.2 Connection to the E-Puck

Before sending commands to the e-puck, the function *connect* must be called. This function takes as parameters an instantiation of the class ePicKernel and a string containing the COM port associated with the e-puck. For instance, *epic=connect(epic,'COM15')* opens a connection with the e-puck associated to port COM 15. The command returns a modified instance of the ePicKernel class. If the connection was successful, the command *get(epic,'connectionStatus')* returns 1, otherwise 0.

Once connected, all the interactions will be done with the same robot. It is not possible to connect to more than one e-puck at a time due to Matlab limitations.

To disconnect, the command *epic=disconnect(epic)* should be called. Afterward, Matlab can be connected to another e-puck on another port.

3.3 Sending and Receiving Data

All the interactions with the e-puck, with the exception of the camera, are centralized in one command *epic=update(epic);*. This command updates the values of the sensors and sends the commands to the e-puck.

All the transfers between the e-puck and Matlab are done simultaneously. First, all the commands are sent to the e-puck. Those can be a request for sensor values or a modification of the e-puck's wheel speed. Then, *update* waits for the reception of the requested information and updates the variables of the class epic. Please read section 3.4 to learn how to select which sensor to read and how to control the e-puck.

3.3.1 Unrecommended method

It is also possible to send custom commands to the e-puck using the low level functions to send and receive data. Those exist either in ASCII or in binary but their usage is the same. The difference resides in the way the data are encoded. In ASCII mode, the data sent by the e-puck are formatted as a string. The Matlab command *str2num* has to be called to convert those into a Matlab variable. In binary mode, the data are formatted as they were on the e-puck and spread over different bytes. It is then necessary to reformat the values into their original format. This is generally more difficult to do as it requires a knowledge of the inner programming of the e-puck. The binary mode is nevertheless more advantageous when multiple commands have to be sent simultaneously. In ASCII mode, all commands are sent separately on the bluetooth channel while in binary the are all sent with one transfer only. This has the effect of speeding up the transfers between the e-puck and Matlab.

Sending a custom command is done through the functions *write(epic,data);* or *writeBin(epic,data);* where data contains the commands to send. The first function uses the ASCII mode while the second the binary mode. Those commands are asynchronous and won't wait for the answer of the e-puck. To receive the result of the sent command, the functions *data=read(epic);* or *data=readBin(epic);* have to be called according to the type of writing you did previously. An alternate command to read binary data is *data=readBinSized(epic,n);* where *n* is the size in bytes of the data to be transfered. The function will wait until all the data has been received or for a timeout to occur.

The final command to know is *flush(epic);* which resets the transmission and reception buffer of Matlab. It is useful to call before any communication with the e-puck as Matlab

does not seem to do it by itself. If not called, there are risks that old information will remain in the buffers.

3.4 Commanding the e-puck

As mentioned earlier, the *update* command handles all the information transfers between Matlab and the e-puck. All data sent or received is stored in the class' variables. By default, ePic2 reads the values of the accelerometer, the proximity sensors, the light sensors, the microphones, the motor speeds and the encoders. Those correspond to the basic functionalities of the e-puck.

3.4.1 Gathering information - get

To activate the reading of a sensor, the command *epic=activate(epic, propName);* must be called where *propName* is the name of the sensor to activate. To deactivate a sensor, the command *epic=deactivate(epic, propName);* must be used.

A complementary function, *epic=updateDef(epic, propName, frequency);* allows you to set the update frequency 0:no update (=desactivate), 1:allways update (=activate), 2:update once only.

Once the information has been retrieved, it can be obtained with the function *[val,up]=get(epic,propName);* where *val* contains the requested values and *up* indicates those were refreshed during the last update.

The following table contains the values for *propName* and should be used as Matlab strings.

propName	**Data**	**Supported Commands**
accel	Accelerometers	get, activate, deactivate, updateDef
proxi	Proximity Sensors	get, activate, deactivate, updateDef
light	Light Sensors	get, activate, deactivate, updateDef
micro	Microphones	get, activate, deactivate, updateDef
floor	Floor sensors	get, activate, deactivate, updateDef
speed	Motor Speeds	get, activate, deactivate, updateDef
pos	Encoders	get, activate, deactivate, updateDef
odom	Odometry Position	get, activate, deactivate, updateDef
external	LIS External Sensor Turret	get, activate, deactivate, updateDef
image	Camera Image	get, activate, deactivate, updateDef
custom	Custom Command	get, activate, deactivate, updateDef
connectionState	Connection Status	get

3.4.1.1 Update image The *update* function cannot get an image from the camera of the e-puck. This is due to the long time needed to transfer the image from the robot to the computer and also to the specific processing done on it. To request an image, the command *[epic, mode, sizexy]=updateImage(epic);* must be called. The returned values mode and sizexy are respectively the color mode of the camera and the size of the image. Once done, use the command *[image,up]=get(epic,'image');* to retrieve the stored image from the class.

3.4.1.2 Update odometry ePic2 can compute the odometry of the robot. For that purpose, the function *[epic]=updateOdometry(epic);*. As before, the function*[pos,up]=get(epic,'odom');* returns the position of the robot. The function *epic=reset(epic,'odom');* will reset the internal variables of the odometry. The command *set* can also modify those variables. More details concerning its use have been given above.

3.4.1.3 Custom commands ePic2 supports the use of custom commands. A custom command is a vector containing Sercom instructions that will be sent to the e-puck during the next update. They will be executed at the end of the regular commands. The results can be retrieved with *get(epic,'custom')*. Those will be raw data in a 8 bits format. It can be necessary to convert them to Matlab's format. Many examples of this can be found in the source code of the *update* function and a function *two_complement* receiving two 8 bits values and returning their equivalent in 16 bits is provided with ePic2.

3.4.2 Controlling e-puck - set

The command *epic=set(epic,varargin);* modifies some properties of the e-puck. *varargin* is a set of parameters beginning with the name of the property to modify followed by a vector containing the new values. For instance, to set the speed of the two motors, the command *epic=set(epic,'speed',[100 100]);* must be called. The following table details the supported properties and their arguments.

propName	**Arguments**	**Description**
speed	[right_motor left_motor]	Change the motor speeds
ledOn	[led_number]	Light on the led number led_number
ledOff	[led_number]	Light off the led number led_number
odom	[x y theta]	Set the current position used by the odometry
camMode	[mode]	Set the camera mode (0: grayscale, 1:color)
camSize	[width height]	Set the width and the height of the camera
camZoom	[zoom]	Set the zoom factor (1, 4 or 8)
external	[options]	Select the external sensor and set its options
ledIR	[leds]	Set the leds to light on or off for the 5 leds external sensor
custom	[commands, bytes to receive]	Set a custom command to be executed by the e-puck

3.5 The Filters

It is sometimes interesting to apply some processing on the readings of a sensor. For instance, the calibration of the proximity sensors is necessary to use them. To allow this in a transparent way, some empty filters have already been created. The functions *filter_Accel*, *filter_Light*, *filter_Micro*, *filter_Floor*, *filter_Prox* and *filter_image2* are called every time a reading is done

on respectively the accelerometer, the light sensors, the microphones, the proximity sensors and the camera. They receive as argument the data and return the filtered values of those. When no filter is implemented, the returned values are the same as the read values. In case of the need for a filter, those functions can be rewritten according to the user needs.

4 The Graphical Interface

The graphical interface has been developed to display the status of the e-puck in real-time. Its programming relies on the above mentioned commands but add some functionalities. The interface is refreshed on a regular interval and can show the values of the sensors in real time, it allows to command the movements of the robots using a joystick-like interface, it displays the camera snapshots before and after filtering and also plot the path of the robot using odometry. All those can be seen in figure 1.

The interface possesses a menu where sensor readings can be activated or deactivated.

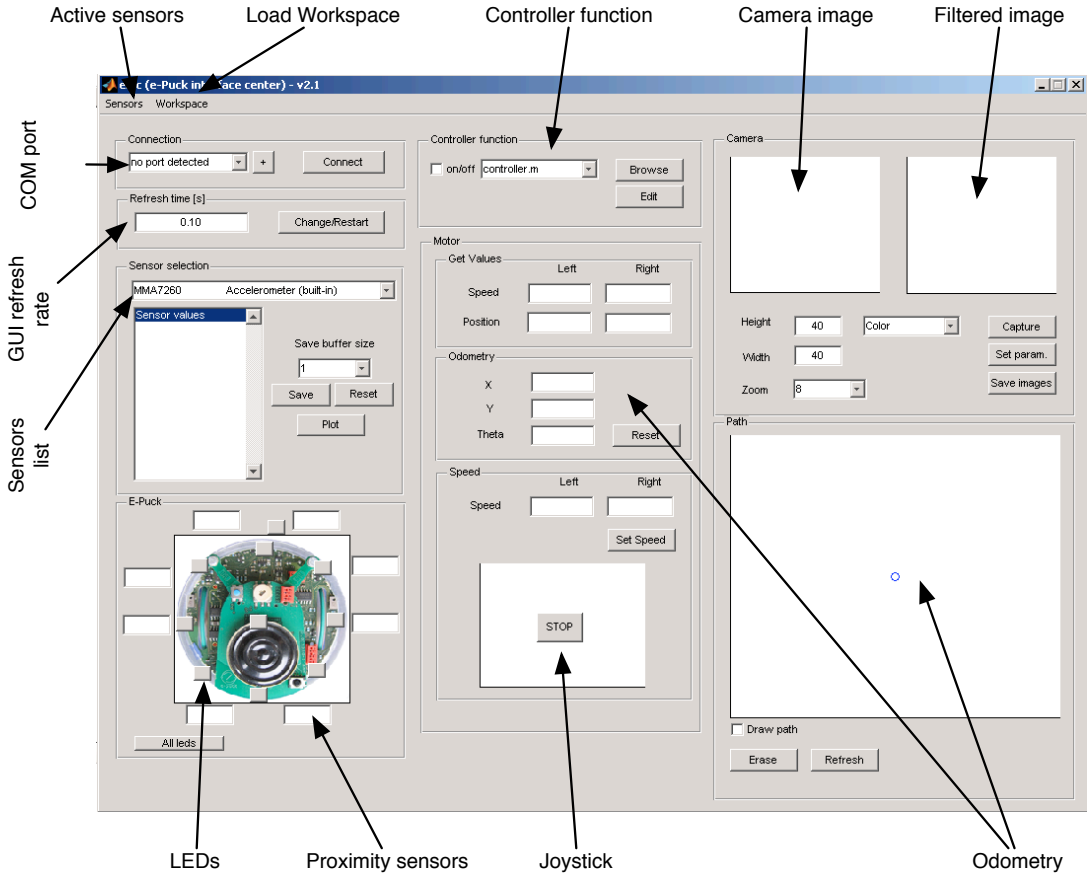

Figure 1: The ePic2.1 Graphical Interface

4.1 The Controller

One feature of the GUI is the implementation of a controller function to command the e-puck. This function is located in the file *controller.m* and is initially empty. Once activated, it can be programmed to add a behavior to the e-puck.

The controller acts as a Matlab script. As such, all the above functions can be used to command the e-puck. The command *update* does not need to be called as it is done automatically by the interface. It is nevertheless necessary to activate the sensors you need for your controller before executing it.

The controller is a finite state machine. It begins in state 1 and executes its initialization. Then it moves to state 2 or more to execute the body of the code. When stopped, it goes to state -1 where it executes the final part of the code before going to state -2. If the controller crashes for no precise reason, the interface detects it and stops its execution.

4.2 Sensor Captures

The interface always saves the values of the currently selected sensor. A box allows to choose how many samples have to be kept. For instance, if 1 is selected, the last value will always be kept in memory. When you want to transfer the saved values to Matlab, press the button *Save* and the last readings will be transfered to Matlab's workspace. In some cases, this button will be disabled and colored in red. It means that the minimum amount of samples requested has not been attained yet. When it will be so, the button will turn green. It is possible to save an undefined number of samples by choosing *Unlimited* as Save buffer size.

The Reset button will empty the Save buffer.

4.3 Driving the Robot

To drive the robot, you have two choices using the interface. You can either set a speed for each motor in the area *Speed* and press the button *Set Speed* or you can use the visual joystick underneath. The white zone of the joystick allows you to set the motor speeds using a more intuitive way. For instance, if you click the top middle of the white zone, the e-puck will run at maximum speed in straight direction. If you press a bit on the right, it will start to turn right. If you press below the button *Stop* it will run backward. The *Stop* button is used to stop the robot.

4.4 The Camera

The camera area is composed of two displays. The first is the current view coming from the robot while the one on the right is the same view after filtering. The camera works in two modes: color or gray scale. The height and the width can be chosen but it should be carefully considered as the e-puck is quite limited in memory. The value 40x40 is an adequate value. The zoom factor describes a zoom out from the center of the image. By choosing 1, the image will be very neat but very focused while 8 gives a wider view with a reduced quality. The number that can be chosen are 1, 2, 4 and 8. After any modification of those values, the button *Set param.* must be pressed to send them to the e-puck. When the button *Capture* is pressed, a new snapshot is taken.

5 Known issues

The initial connection can take quite a long time sometimes. This is due to the Windows driver of some computers. Don't despair and wait. It will connect finally. When closing the connection, it is possible that Matlab crashes and has to be killed. This is also due to the Windows driver and occur only on some machines.

6 Credits

ePic2 was developed at the Laboratory of Intelligent Systems located in the Ecole Polytechnique Fédérale de Lausanne in Switzerland by Yannick Weibel and Julien Hubert.