Use of Python and Phoenix-M interface in Robotics

Shubham Chakraborty

Editor(Physics- Higher Academics), Vikas Publishing House Pvt Ltd
E-28, Sector-8, NOIDA, UP
email: chakraborty.shuvam@gmail.com, +91-9999192634

If at first an idea doesn't look absurd, it has no future- Sir Albert Einstein

Abstract: In this paper I will show how to use Python programming with a computer interface such as Phoenix-M [1] to drive simple robots. In my quest towards Artificial Intelligence(AI) I am experimenting with a lot of different possibilities in Robotics. This one will try to mimic the working of a simple insect's nervous system using hard wiring and some minimal software usage. This is the precursor to my advanced robotics and AI integration where I plan to use a new paradigm of AI based on Machine Learning and Self Consciouness via Knowledge Feedback and Update Process.

Index Terms: Python, Phoenix-M, Robotics.

1. Introduction

This paper is neither path-breaking nor is it about some great inventions. It is about a humble Robot which will showcase a simple thing that how easy is it to use python for works like robotics. Robotics is a field which has shown tremendous promise and has been equally dissappointing when it comes to deliver the promises. Modern day robots are driven by some seriously sophisticated software in conjunction with equally sophisticated mechatronics. In this paper we will see how we can use the Python programming language and Phoenix computer interface to drive simple robots. I have used some very minimal Python programming which can drive the Phoenix interface to link a computer with the robot. For a robot, I have modified a remote controlled toy car and integrated it with Logic and electronic circuits and Phoenix interface. But before I write something about it let us first discuss about the Phoenix interface,.

2. PHOENIX-M

Phoenix-M[1] is a micro-controller based computer interface which has been developed by Inter-University Accelerator Centre.[2] This simple yet powerful interface has been created to bridge the gap between inexpensive computers and highly expensive computer interfaces, that can be found in sophisticated laboratories only, to garner data and analyse them.

[1] **P**hysics with **H**ome-made **E**quipment and **I**nnovative **E**xperiments

[2] Formerly Nuclear Science Center, Delhi. IUAC is an autonomous research institute of University Grants Commission, India,providing particle accelerator based research facilities to the Universities.

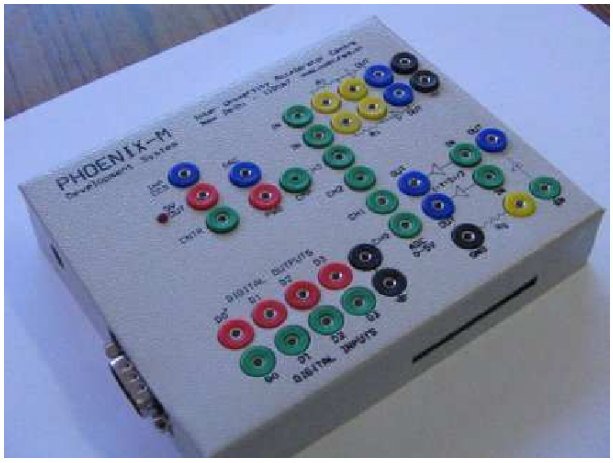

Fig. 1. Phoenix Interface[4]

Phoenix provides microsecond level accuracy for timing measurements. Collecting data from the sensor elements and controlling the different parameters of the experiments from the PC is one of the features of Phoenix-M. This is achieved by loading the required software[3] into the micro-controller. This enables the students to use it as a general purpose micro-controller development kit as well as help them design stand-alone projects. The interface comes with a CD which contains all the required software to drive the micro-controller [4]. The python library with which the controller can be accessed is known as `phm.py` and can be found in the software repositories specially made for Phoenix.

3. Lets get started

The main aim of this experiment is two fold.

1) To create a simple robotic device which will mimic the way a **Foraging ant[3]** searches for food by tracing chemical signature of food.
2) To use Phoenix-M[5] and Python in sync, to create this robot, and put forward the versatility of Python programming.

This toy robot is inspired by the actions of Foraging ants[3]. It has to start from its home and wade through a series of obstacles, to reach its desired destination. This is very similar to the actions of Foraging ants[3] which go out in search of food. If food is nearby, it gets the chemical signature and starts following it, while overcoming all types of obstacles it finda in its way. This will be done using Micro-controller based controls of Phoenix-M interface.

When a Foraging ant[6] goes out in search of food, it leaves behind a trail of **Pheromones[2]**[7]. Once food has been found, it traces backs its pheromonal trail and reaches its home. *In our case we have a toy robot which will trace a light signal* [8].

[3]The software is usually **Python**
[4]Atmega16
[5]Dr Ajith of IUAC will be giving a lecture on Phoenix-M
[6]The only difference is that initially Fooraging ants wander a bit randomly till they reach their destination and then their path becomes more stable. In our case that will not happen
[7]A pheromone is a secreted or excreted chemical factor that triggers a social response in members of the same species.
[8]Light signal is our substitute of chemical signature of food

3.1. Overview of variables and definitions

Before I start explaining everything let us first decide on the variable conventions. So I will be using the following variables which have some specific meanings. Here is a list of variables and their meanings.

Table 1: Variables and their meanings.

Seek	The eye of the robot seeks the light signal.
P_L	Left pressure sensor.
P_R	Right pressure sensor.
W_L	Left wheel.
W_R	Right wheel.
$S_{1,2,3,4}$	Switches 1 to 4.
1	Forward Movement
0	No Movement
$\overline{1}$	Backward Movement

3.2. Algorithm

First we have to develop an **Algorithm**. The algorithm is a step by step process which a certain system follows to achieve its goals. In our case the Robot has to first seek out the light source. The moment it gets hold of the light source it moves towards it. In case it encounters any obstacle, its pressure sensors immediately signals the micro-controller. The decision is taken according to the side which encounters the obstacle.

Table 2: Algorithm followed by the Robot

Step 1	Start
Step 2	Seek Light Source
Step 3	If (Seek Light ==1) Then Move ahead
Step 4	If (Obstacle==1) Then
Step 5	If(Obstacle == Right) Then Move Clockwise / Back
Step 6	If(Obstacle == Left) Then Move Anti-Clockwise / Back
Step 7	Else If(Obstacle == Left == Right) Move Back and stop for 2 seconds
Step 8	Repeat Step 2 to 7 till destination reached
Step 9	Stop

One thing to note is that 1 *means Forward movement,* 0 *means No movement and* $\overline{1}$ *means Backward movement*

3.3. Control Sequences

Now lets see the **Movement Sequence of the Robot**. The movement sequence lets us determine the way the robot should react to an incoming signal stimulus.

Table 3: Movement Sequence

Move Ahead	W_L = W_R = 1
Move Clockwise/Back	$W_L = 0$ and $W_R = \overline{1}$
Move Anti-Clockwise/Back	$W_L = \overline{1}$ and $W_R = 0$
Seek	$W_L = 1$ and $W_R = \overline{1}$

3.4. Seek and Move Procedure

Table 4: Truth table for Seek and (Respective) Move Procedure

Seek	P_L	P_R	W_L	W_R
1	0	0	1	$\overline{1}$
0	0	0	1	1
0	1	0	0	$\overline{1}$
0	0	1	$\overline{1}$	0
0	1	1	$\overline{1}$	1

Table 5: Truth table for Move Procedure and Switch Combinations

W_L	W_R	S_1	S_2	S_3	S_4
0	0	0	0	0	0
1	$\overline{1}$	1	0	1	0
1	1	0	1	1	0
0	$\overline{1}$	0	0	0	1
$\overline{1}$	0	1	0	0	0
$\overline{1}$	1	0	1	0	1

Now that we have seen the various procedures that are required to make the robot work, we will now concentrate on the second most important part of the experiment, Hardware and Software. There are essentially two hardware components that we will use in sync; **Phoenix-M** interface and the **Robot** whereas the software will be a combination of Python code which will make the robot move and the another Python code to connect to the Phoenix-M interface and make its micro-processor respond.

The given figure is just an outline of the Robot. WL and WR are the two wheels which run independant of each other. PL and PR are the two electro-mechanical pressure sensors which sense whether there is an obstruction in the path or not. If there is an obstruction, the pressure sensors complete a circuit which grounds the digital input channels of Phoenix-M interface. Similarly there is another sensor, $Seek$ which acts like the eye of the Robot. It is an LDR(Light Dependant Resistor) which is attached to the base of an n-p-n transistor such that the transistor acts like a switch. The switch is used to operate a relay circuit. The output of the relay is taken as Logic High (+5V) or Logic Low(0V). Apart from this it also houses a battery pack, a PCB on which the sensors are mounted and relay switches which help in the passage of signals. Below you will see the circuit diagrams of the components and how they can be integrated with the Phoenix-M interface.

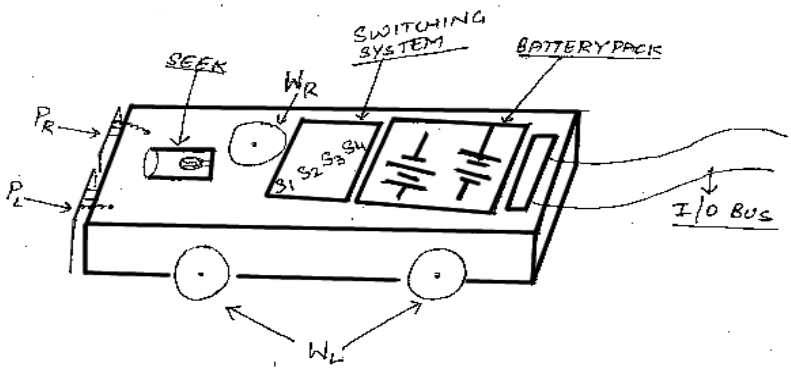

Fig. 2. Outline of Robot

3.5. Bit combinations

The Phoenix interface has its own way to understand and decipher the input and output signals through its channels. These Bit combinations need to be understood before we can start to build our Python code based on these combinations.

Table 6: Input channels on Phoenix-Mand their Bit combinations

S.No.	Input Channels	Input Signal	Bit Conventions
1	D_0	$Seek$	1
2	D_1	P_L	2
3	D_2	P_R	4

Table 7: Output channels on Phoenix-Mand their Bit combinations

S.No.	Output Channels	Output Signal	Bit Conventions
1	D_0	S_4	1
2	D_1	S_3	2
3	D_2	S_2	4
4	D_3	S_1	8

3.5.1. What does the above mean?

In the output channels if I want to make both the wheel move forward then according to Table 5, S_3 and S_2 should go high. Since S_3 and S_2 are connected to D_1 and D_2 respectively, then the output bit combination should be 0110 which in decimal system is 6. The following two tables will show you the bit combinations.

Table 8: Modified Truth table for Seek Procedure with Bit combination

Seek	P_L	P_R	W_L	W_R	Bit combination
1	0	0	1	$\overline{1}$	15
0	0	0	1	1	14
0	1	0	0	$\overline{1}$	10
0	0	1	$\overline{1}$	0	12
0	1	1	$\overline{1}$	1	08

Table 9: Modified Truth table for Move Procedure, Switch and Bit Combimations

W_L	W_R	S_1	S_2	S_3	S_4	Bit combination
0	0	0	0	0	0	00
1	$\overline{1}$	1	0	1	0	10
1	1	0	1	1	0	06
0	$\overline{1}$	0	0	0	1	01
$\overline{1}$	0	1	0	0	0	08
$\overline{1}$	1	0	1	0	1	05

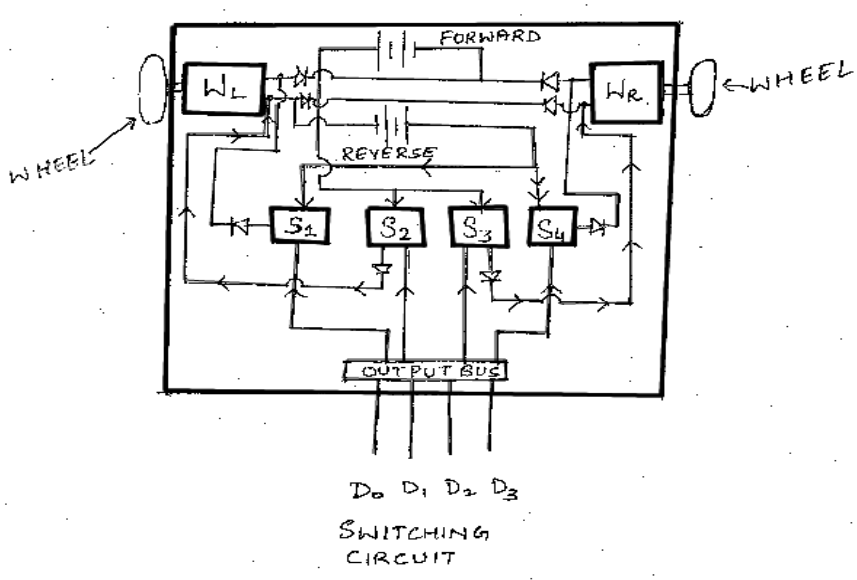

Fig. 3. Circuit Diagram of Switching Circuit

4. Python Codes to ineract with Phoenix-M

4.1. Example 1.

```
import phm, time
p = phm.phm()
while 1:
  p.write_outputs(4)
  time.sleep(10)
  p.write_outputs(0)
  time.sleep(10)
```

A line by line explanation of the Example 1.
Line 1: Loads **phm** module from Phoenix library and **time** module from standard Python library
Line 2: Loads the function `phm()` onto the compiler
Line 3: Starts an infinite loop
Line 4: Outputs a digital signal to BIT Combination 4, which means it will set D_2 high.
Line 5: Stops the execution of the code for 10 seconds
Line 6: Outputs a digital signal to BIT Combination 0, which means all the output channels will set to low
Line 7: Stops execution of the code for another 10 seconds.

As you can see that a simple code like the one given above can actually be used to create a timer device. So instead of changing the circuit, all we need to do is change the code with perfect results. As can be seen, Python makes life a lot easier when it comes to dealing with micro-controllers.

Now let us look at another example in which a specific input gives rise to an output. The `while 1:` command keeps the processor on its toes. It keeps looking for an input from the interface.

4.2. Example 2.

```
import phm, time
p = phm.phm()
while 1:
  p.read_inputs()
  if(p.read_inputs() == 7):
    p.write_outputs(8)
  else:
    time.sleep(10)
```

5. Python code to drive the Robot interface

Let us now look at the code that will drive our Robot across the obstacles to its destination. The codes, though may look very simple and kiddish, are extremely powerfull and flexible and can easily drive the Robot around.

Robot code

```
import phm, time
p=phm.phm()
while 1:
   x = p.read_inputs()
   if (x == 15):
      p.write_outputs(10)
      continue
  elif(x ==14):
     p.write_outputs(06)
     continue
  elif(x ==10):
     p.write_outputs(08)
     continue
  elif(x ==12):
     p.write_outputs(01)
     continue
  elif(x ==08):
     p.write_outputs(09)
     time.sleep(2)
     p.write_outputs(05)
     continue
  else :
     p.write_outputs(06)
```

Table 10: Analysis of the entire code

Code	Function
`x = p.read_inputs()`	Reads input Bit combination from the input channels
`if (x == 15):`	P_L = P_R = 0. $Seek = 1$. Search for light
`p.write_outputs(10)`	Move counter-clockwise
`elif(x ==14):`	$P_L = P_R = 0$. $Seek = 0$. Move towards light
`p.write_outputs(06)`	Move forward
`elif(x ==10):`	$P_L = Seek = 0$, $P_R = 1$
`p.write_outputs(08)`	Move Back/Counter-Clockwise
`elif(x ==12):`	$Seek = P_L = 1$ and $P_R = 0$
`p.write_outputs(01)`	Move Back/Clockwise
`elif(x ==08):`	$Seek = P_L = P_R = 1$
`p.write_outputs(09)`	Move Back
`time.sleep(2)`	Wait 2 seconds
`p.write_outputs(05)`	Move Clockwise
`continue`	After every command, process begins again
`else :`	If nothing matches
`p.write_outputs(06)`	Move forward

The above code, in sync with the micro-controller and the logic (and electronic) circuits, can

easily drive the Robot to its desired destination.

6. What Next!!

Let us now modify the above code for the Robot and get something else done by it. Till now we were working with a Robot which had to bump into something to figure out that it has ran into obstacle. But wouldn't it be nice if it doesn't have to run into an obstacle, just go across it. This can be easily done by modifying the Robot hardware a bit. instead of a light sensitive seeker we will have to use a sonar system. Such systems are easily available in the market. All that has to be done is to mount the sonar system in place of the LDR (Light Dependant Resistor - Seeker) and integrate it with the interface just the way we have done it before. The code will also change a bit. Following is the code and its analysis.

Robot code with Sonar

```
import phm, time
p=phm.phm()
while 1:
   x = p.read_inputs()
   p.write_outputs(2)
   y=p.clr2rtime(0,0)
   distance = (33000*y)/2
  if (distance <5):
    p.write_outputs(09)
    time.sleep(2)
    p.write_outputs(05)
    continue
  elif(x==15):
    p.write_outputs(10)
    continue
  else:
    continue
```

7. Conclusions

What we can conclude from this paper is that Python can actually make the life a lot easier when it comes to using micro-controllers and hence can be a big boon for Robotics enthusiasts. Infact the flexibility and ease with which Python is being implemented has made it a language to reckon with. As we have seen, it can easily interact with the micro-controllers using a simple interface like Phoenix-M. Just by the mere tweakingofpython codes it becomes pretty simple to control the Robots. It seems that more and more experiments will lead to some fascinating research in this particular subject.

Appendix 1: The brief outline of Phoenix-M interface[1]

On the front panel you will find several 2mm banana sockets with different colors. Their functions are briefly explained below.

1. 5V OUT - This is a regulated 5V power supply that can be used for powering external circuits. It can deliver only upto 100mA current , which is derived from the 9V unregulated DC supply from the adapter.
2. Digital outputs - four RED sockets at the lower left corner . The socket marked D0* is buffered with a transistor; it can be used to drive 5V relay coils. The logic HIGH output on D0 will be about 4.57V whereas on D1, D2, D3 it will be about 5.0V. D0 should not be used in applications involving precise timing of less than a few milli seconds.
3. Digital inputs - four GREEN sockets at the lower left corner. It might sometimes be necessary to connect analog outputs swinging between -5V to +5V to the digital inputs. In this case, you MUST use a 1K resistor in series between your analog output and the digital input pin.
4. ADC inputs - four GREEN sockets marked CH0 to CH3
5. PWG - Programmable Waveform Generator
6. DAC - 8 bit Digital to Analog Converter output
7. CMP - Analog Comparator negative input, the positive input is tied to the internal 1.23 V reference.
8. CNTR - Digital frequency counter (only for 0 to 5V pulses)
9. 1 mA CCS - Constant Current Source, BLUE Socket, mainly for Re- sistance Temperature Detectors, RTD.
10. Two variable gain inverting amplifiers, GREEN sockets marked IN and BLUE sockets marked OUT with YELLOW sockets in between to insert resistors. The amplifiers are built using TL084 Op-Amps and have a certain offset which has to be measured by grounding the input and accounted for when making precise measurements.
11. One variable gain non-inverting amplifier. This is located on the bot- tom right corner of the front panel. The gain can be programmed by connecting appropriate resistors from the Yellow socket to ground.
12. Two offset amplifiers to convert -5V to +5V signals to 0 to 5V signals. This is required since our ADC can only take 0 to 5V input range. For digitizing signals swinging between -5V to +5V we need to convert them first to 0 to 5V range. Input is GREEN and output is BLUE.

Acknowledgements

I would like to thank the creators of Phoenix-M, without whom neither this interface nor this paper would have seen the light of the day.

References

[1] Ajith Kumar B.P, Pramode C.E. ”Innovative Experiments using Phoenix”, Version 1, 2006.
[2] http://en.wikipedia.org/wiki/Pheromone
[3] http://en.wikipedia.org/wiki/Ant_colony_optimization-Rank-based_ant_system_.28ASrank.29
[4] http://www.iuac.res.in/ elab/phoenix/