The Software Framework

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Robotic Systems

The Framework

- During the course we write and use a Python-based software framework
- It includes:
  - Models of real systems (carts, arms, motors, etc.)
  - Standard control algorithms
  - Data manipulation and visualisation
  - Graphical interfaces (1D, 2D and 3D)
- It can be downloaded (and updated) from
  `https://github.com/corradosantoro/RoboticSystems`

Directories Organisation

- `lib/models/` models of physical systems
- `lib/controllers/` control algorithms
- `lib/gui/` graphical user interfaces
- `lib/data/` classes for data reading from file and output plotting
- `tests/` various test programs

Models

- A physical system must be modeled using a class that includes:
  - Model state variables as attributes
  - Model behaviour implemented in method **evaluate(delta_t, input)**

```
class Cart:


    def __init__(self, _mass, _friction):
        self.M = _mass
        self.b = _friction
        self.speed = 0
        self.position = 0


    def evaluate(self, delta_t, _force):
        new_speed = (1 - self.b * delta_t / self.M) * self.speed + \
                    delta_t * _force / self.M
        new_position = self.position + self.speed * delta_t

        self.speed = new_speed
        self.position = new_position
```

The Framework

Robots

- To simulate a robotic system a subclass of **`RoboticSystem`** must be written that includes:
  - The various pieces of the robot, including physical systems, control algorithms, etc.
  - A call to the ctor of the super-class passing the sampling interval
  - The method **`run()`**, invoked as a callback for each time interval, implementing the behaviour of the overall robot
  - The method **`get_pose()`** returning the pose (position) of the robot
  - The method **`get_speed()`** returning the speed(s) of the robot
- A **`RoboticSystem`** object provides two attributes:
  - **`t`**, the current simulation time
  - **`delta_t`**, the simulation time interval

RoboticSystem Example

```
from models.cart import *
from models.robot import *

class CartRobot(RoboticSystem):

    def __init__(self):
        super().__init__(1e-3) # delta_t = 1e-3
        # Mass = 1kg
        # friction = 0.8
        self.cart = Cart(1, 0.8)

    def run(self):
        self.cart.evaluate(self.delta_t, 2) # 2 Newton
        return True

    def get_pose(self):
        return self.cart.position

    def get_speed(self):
        return self.cart.speed
```

The Framework

GUI

- The GUI is a Qt5 class (generally called **MainWindow**) the implements:
  - The visualisation of our environment + the robotic system
  - The engine that calls the **run()** method of our robotic system for each time interval
- It must be instantiated by passing a **RoboticSystem** object in the ctor

```
from models.cart import *
from models.robot import *
from gui.gui_1d import *

from PyQt5.QtWidgets import QApplication


class CartRobot(RoboticSystem):
    ...

cart_robot = CartRobot()
app = QApplication(sys.argv)
ex = MainWindow(cart_robot)
sys.exit(app.exec_())
```

The Framework

FileReader

- Inputs to the system can be hard-coded or read from a data file; in the second case, a **FileReader**) can be used
- It provides the following methods:
  - **FileReader(filename)**, the object must be created by passing the data file name in the ctor
  - **FileReader.load()**, it loads and interpretes data from the file
  - **FileReader.get_vars(t, varlist)**, it retrives data of the variables given in varlist according to current time t

Data File

```
t , F
0.0, 10.0
1.0, 0.0
```

- First line: the names of variables (comma-separated string, the first is always the time)
- Second and subsequent lines: the values of the variables given a certain (absolute) time (comma-separated floats, the first value is always the time)

The Framework

Using the FileReader

```
...
from data.readers import *
...

class CartSystem(RoboticSystem):

    def __init__(self, filename):
        super().__init__(1e-3) # delta_t = 1e-3
        # Mass = 1kg
        # friction = 0.8
        self.cart = Cart(1, 0.8)
        self.datafile = FileReader(filename)
        self.datafile.load()

    def run(self):
        [ F ] = self.datafile.get_vars(self.t, [ 'F' ])
        self.cart.evaluate(self.delta_t, F)
        return True

...
```

DataPlotter

- Data used in simulation can be plotted in charts using a **DataPlotter** object; it is a data collector able to draw charts
- It offers the following methods:
  - **DataPlotter.add(varname, varvalue)**, adds a new value in the trend of a variable called varname
  - **DataPlotter.plot(x, y)**, plots data using the specifications provided in parameters x and y:
    - **x = [x_var, x_label]**, x_var is the variable whose trend can be used as X axis, x_label is the label shown in X axis
    - **y = [[y_var1, y_label1], [y_var2, y_label2] ...]**, y_varN is the variable whose trend has to be shown, y_labelN is the label shown in the legend
  - **DataPlotter.show()**, shows the figures with the plots

Using the DataPlotter

```
...
from data.readers import *
from data.plot import *
...

class CartSystem(RoboticSystem):

    def __init__(self, filename):
        super().__init__(1e-3) # delta_t = 1e-3
        # Mass = 1kg
        # friction = 0.8
        self.cart = Cart(1, 0.8)
        self.datafile = FileReader(filename)
        self.datafile.load()
        self.plotter = DataPlotter()

    def run(self):
        [ F ] = self.datafile.get_vars(self.t, [ 'F' ])
        self.cart.evaluate(self.delta_t, F)
        # gather data into plotter variables
        self.plotter.add('t', self.t)
        self.plotter.add('F', F)
        self.plotter.add('v', self.get_speed())
        self.plotter.add('p', self.get_pose())
        if self.t >= 4: # after 4 seconds plot data and stop simulation
            # prepare a figure with plots for force and speed
            self.plotter.plot(['t', 'time'], [ [ 'F', 'Force' ],
                                               [ 'v', 'Speed' ] ])
            # show the plots
            self.plotter.show()
            return False
        else:
            return True
```

The Software Framework

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Robotic Systems