Petri Net based Modelling and Control of Humanoid Robots

Reggie Davidrajuh

Electrical and Computer Engineering
University of Stavanger
Stavanger, Norway
E-mail: reggie.davidrajuh@stud.uis.no

***Abstract* — This paper presents an approach to model a humanoid robot as a discrete event system; the approach is based on Petri Nets. Firstly, this paper proposes representation of a human body with a system of ten interrelated links. Secondly, each link is modelled as a Petri net module resulting in ten independent modules. Thirdly, this paper proposes use of a command-dispatcher module to coordinate all the other modules; thus, the command-dispatcher module acts as the brain of the robot. Finally, as shown in the final section, the modular Petri Net based model can be easily used for model-based control of a humanoid robot.**

Keywords - Humanoid Robot; Petri Nets; GPenSIM; Range of Motion.

I. INTRODUCTION

In this paper, an approach is presented to model a humanoid robot as a discrete event system. Humanoid robots are the robots, which are replication of humans having a head, torso, two arms, and two legs [1]. The modeling approach presented in this paper is based on Petri Nets. Petri net is a powerful framework for modeling of discrete event systems; since this work sees a humanoid robot as a discrete event system, Petri Net suits well to model the discrete dynamics of a humanoid.

Literature study reveals modeling of humanoid robots using various methodologies, approaches, and techniques. Reference [1] uses modeling and simulation of humanoid for swarm robotics, where a large number of robots are involved in coordinated acts. Reference [2] uses geometrical and inertial properties of the various joints and links of a humanoid robot for parametric modeling of kinematics. Reference [3] uses real-time gait patterns and gait planning of a humanoid robot, using the Local Axis Gait algorithm; reference [4] focuses on postures for different situation in sports and everyday life, by modeling humanoid robot for posture stabilization. Reference [5] models a humanoid for simulating motions of humans in a "futuristic distributed tele-immersive collaboration system", where various teams of performers perform at geographically distributed places and their performances are presented on live TV program as the teams are performing on the same place at the same time.

However, to our best knowledge, this paper is the first attempt for modeling, simulation, and control of a humanoid robot using a *modular Petri Net* based approach; the modules represent the 10 links of a human body.

In this paper: A short literature study is presented in section-II. Section-III introduces the basics of forward kinematics of rigid human body in 3D space. Section-IV and V present a new modular approach based on Petri Nets for modeling of humanoid robots; section-IV presents the main modules of the system and section-V presents the submodules. Section-VI presents the implementation details for real-time control of a humanoid robot. The implementation is done with GPenSIM. Section-V presents some pointers for further work.

II. RELEVANT WORKS

Literature study reveals many works on modeling of humanoid robots with Petri nets. Reference [6] presents supervisory control based model for motion planning of humanoid robots; supervisory control is a control technique based on Petri nets. In the reference [6], a modular state space model is introduced to eliminate the complexities generated by the many sensors. The modular state space model is composed of two level, where the upper level generates optimal sequences of actions and the lower level uses the actions to control and monitor the actuators and sensors [6]. Reference [7] also uses Petri net based supervisory control for modeling and control of a humanoid robot. In this work, a 3-layered hierarchical architecture is used, for planning (highest layer), execution, and supervision (lowest layer) of robotic actions. The knowledge bases, databases, and rules are connected with the highest layer, and robotic hardware (sensors and actuators) and connected with the lowest layer [7].

Reference [8] is about transferring knowledge from humans to humanoids, by the Programming by Demonstration (PbD) framework. To realize the PbD framework, a three-layered control architecture is proposed, which consists of task planning, task coordination, and task execution layers. The task coordination layer is based on Petri nets [8]. Reference [9] provides an approach for temporal coordination of a humanoid robot; for temporal coordination, an extended Petri net is used. In this extended Petri net, every place and transition can be assigned with specific functions, callback functions with places and trigger functions with transitions. Whenever a place is loaded with tokens, the callback function associated with this place (if available) will be automatically executed. Similarly, whenever a transition becomes enabled, the trigger function associated with it will be automatically executed. Hence, the

logic for temporal coordination is realized jointly by the static Petri net structure and the functions associated with the places and the transitions. The work uses the extended Petri net approach for modeling as well as for control of the robots.

Reference [10] presents a two-phase approach for modeling and control of robots. In the first phase, the modeling and simulations are done with Petri Nets. In the second phase, the Petri Net model is converted into Java code for implementation.

A. *Why is this paper unique?*

Similar to many other works, this paper also uses a three-layered based architecture. However, unlike the other works, the three layers are based on human anatomy: a module representing the head and named as the "control & command dispatcher" sits on the top layer. The human body is modeled as a 10-links based discrete model, and the modules representing these links are placed in the second layer. Finally, in the lowest-level third layer sits the modules for realizing the motions, the alpha, beta, and gamma modules, as explained in the following sections. In addition, this paper proposes the use of a new tool called GPenSIM, not only for modeling and simulation but also for control (implementation) of humanoid robots.

III. Human Body as an Aggregation of Ten Links

This section summarizes the ideas presented in the works [5], [11], and [12], on human body as an aggregation of ten interrelated links, where each link is subjected to a limited range of motion (ROM) in 3D space.

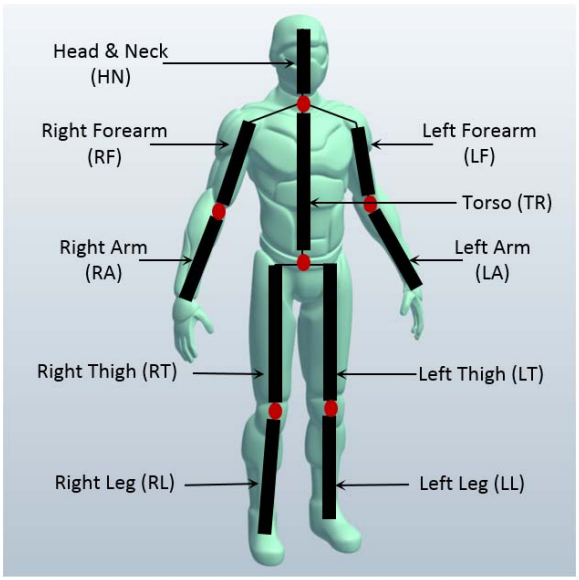


Figure 1. Human body as an aggregation of ten interrelated links.

A. *Simplified model of human body*

The simplified model of a human body as an aggregation of ten interrelated links is shown the figure-1. In this model, the smaller links such as fingers and toes are abstracted away as they do not contribute to the general overall movement of humans. In addition, neck and head are combined for simplicity. The model shown in the figure-1 can be considered as a discrete event system, if we limit the motion of the links into discrete steps.

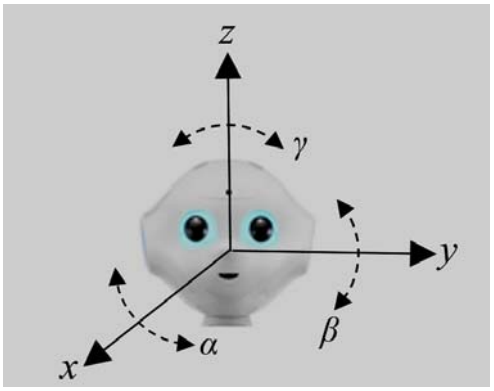


Figure 2. A three dimensional Cartesian coordinate system, for the angular displacements along axes x, y and z.

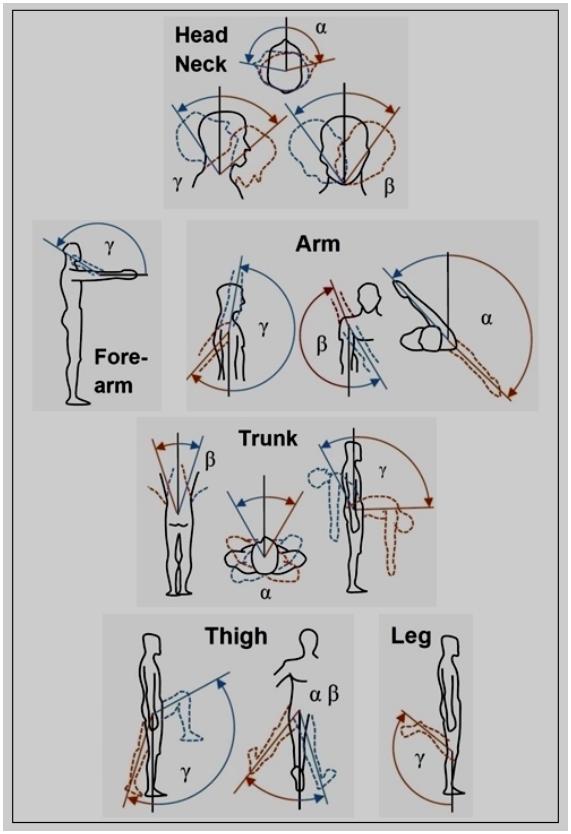


Figure 3. The normal range of motion (ROM) of the ten links in 3D space [5, 11, and 12]; ROM of each link is represented by the angles α, β and γ.

The simplified model of the human body as shown in the figure-1 is a rigid body, as the interrelated links are fixed either at one or two points, making the solid body free from deformation (the distance between any two given points of

the rigid body do not change while on motion). In order to facilitate the motion of rigid human body, the 3D workspace is represented by the axes *x* and *y* (on the horizontal plane) and by the *z*-axis as the vertical line. In addition, the angular displacements are represented by α, β and γ on the *x*, *y*, and *z*-axis, respectively, as shown in the figure-2.

B. Normal range of motion (ROM) of a rigid human body

Taking the whole body as rigid, the normal range of motion (ROM) of the ten links are shown in the figure-3. Based on the information depicted in the figure-3, Table-1 presents the ROM of the ten links. Table-I also shows that ROM of left arm and right arm are the mirror image of each other on the *x* and *y* axes. This is same for the left and right thighs. In addition, forearms and legs have only one degree of displacement, along the z-axis.

TABLE I. RANGE OF MOTION (ROM) OF THE TEN LINKS [5, 11, 12]

Links	A	β	γ
Head & Neck (HN)	(-70, 70)	(-63, 63)	(-60, 30)
Forearms (LF/RF)	(0, 0)	(0, 0)	(0, 150)
Left Arm (LA)	(-30, 130)	(-40, 170)	(-40, 170)
Right Arm (RA)	(-130, 30)	(-170, 40)	(-40, 170)
Torso (TR)	(-30, 30)	(-40, 40)	(-90, 30)
Left Thigh (LT)	(-50, 45)	(-30, 45)	(-15, 90)
Right Thigh (RT)	(-45, 50)	(-45, 30)	(-15, 90)
Legs (LL/RL)	(0, 0)	(0, 0)	(-145, 0)

IV. THE PETRI NET MODEL

The simplified model of the human body shown in the figure-1 is used as the starting point in developing a modular Petri Net model of a humanoid. As the figures 1 and 3 depict, the Petri net model should possess at least ten modules, one for each link. In addition, each module must facilitate the motion of the link on 3D space, along the *x*, *y*, and *z*-axis, indicating the current values of α, β and γ at any point of time. In addition, in order to achieve more realistic simulation of a human body, each module must limit the motion of the link within the respective ROM.

It will be very difficult (if not impossible) to impose the ROM limitation on a Petri Net module, using basic (or P/T) Petri Nets. This is because P/T Petri Nets lacks the modeling power that is needed to solve complex problems [G. Ciardo]. Hence, this paper proposes a hybrid approach, showing only a part of the logic on the Petri Net and the rest on the run-time software. This hybrid approach of showing part of the logic on the Petri net model ("hard-wiring") and coding the rest of the logic on the run-time software ("soft-coding") is a well-proven approach that not only paves simple Petri Net models, but also the other benefits such as fast model building, simple debugging, and easy extension [AOPN]. In the following subsections, the Petri Net model a simple humanoid is built step-by-step, using a modular approach.

A. The modules of the system

The model of a humanoid robot consists of individual modules that get commands from a superior through message passing, and the modules let know the status (current values of the angles α, β and γ) through global variables ("pigeon holes"). As shown in the figure-4, the commands are generated by the module "commands generator and dispatcher", which generates the commands and places the commands in a common buffer (a Petri Net place) **pCommandsBuffer** (**pCommandsBuffer** is not shown in the figure-4).

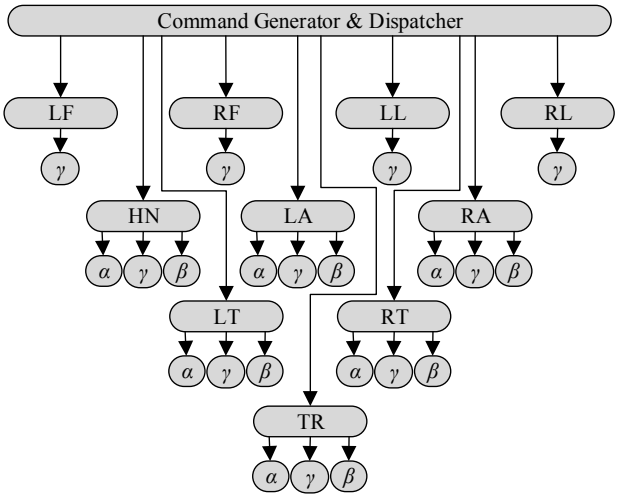


Figure 4. Generation and delivery of commands for the movements of the ten links.

In each of these commands, there is a field called 'module' that clearly identifies the targeted module, and another field called 'axis' that identifies the angle that is subject to change.

command no.:	1
module:	HN
axis:	*Z*
action:	Add
degrees:	5
start-at:	0
duration:	60

command no.:	2
module:	TR
axis:	*Y*
action:	Goto
degrees:	-40
start-at:	60
duration:	60

command no.:	3
module:	HN
axis:	*x*
action:	goto
degrees:	30
start-at:	120
duration:	60

command no.:	4
module:	HN
axis:	*x*
action:	goto
degrees:	-30
start-at:	180
duration:	60

Figure 5. Four sample commands and their fields.

For example, in the four commands that are shown in the figure-5, the first command is intended for the HN module, where the head is supposed to move around the z-axis. Whereas, the second command is intended for the TR module, where the TR is to be changed on the y-axis.

From the common buffer **pCommandsBuffer**, individual modules representing the ten links will pull the commands that are directed for them. After pulling the command, the individual modules will interpret the command and then pass it to the relevant submodules that are responsible for changing the angles α, β or γ. For example, after pulling the first command (shown in the figure-5), the HN module will interpret it (meaning, the angle γ should be increased by 5 degrees) and then pass it to its submodule γ.

There are two types of actions coded in a command: the 'add' action demands that the angle specified by the field 'degrees' should be added to the current value; the 'goto' action demands that the relevant angle should be set the specified degrees, immaterial of its current value. For example, the first command shown in the figure-5 expects that the current value of γ should be increased by 5 degrees ('add'); whereas the second commands expects the torso is changed to a position where the angle β becomes -40 degrees, from whatever the current value be ('goto').

The figure-5 also shows that the commands have two other fields as well: 'start-at' and 'duration'. The field 'start-at' dictates the time at which the action should be started; if the 'start-at' value is zero, then the action must commence at once. For any other positive real number for 'start-up', the action must wait until the clock has passed that value. E.g. in the third command, the action must start when the clock is at 120 TU. Finally, 'duration' represents the length of time during which the action should last. E.g. the fourth command indicates that the action of moving α to -30 degrees from whatever current value, should take a duration of 60 TU.

B. *The Head & Neck module*

The Petri net module for the head & neck (HN module) is shown in the figure-6.

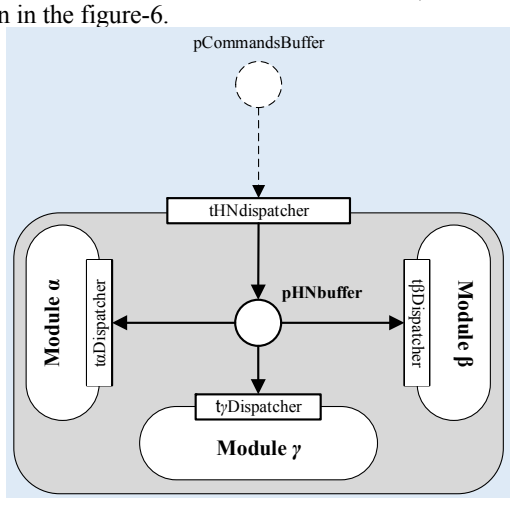


Figure 6. The Head & Neck (HN) module.

Since the motion of the head is determined by all the three angles, namely α, β and γ, the HN module possess the three submodules for changing the angles. The transition **tHNdispatcher** acts as the interface of the module; **tHNdispatcher** pulls the commands from the commands buffer **pCommandsBuffer**. **tHNdispatcher** pulls only the commands that are directed for the HN module, and places these commands into the place **pHNbuffer**, which is positioned in the center of the HN module.

From the place **pHNbuffer,** the submodules that are responsible for changing the individual angles will pull the commands that are directed for them, using the field 'axis' of the commands. The three transitions **tαDispatcher**, **tβDispatcher**, and **tγDispatcher**, are the ones that pull the relevant commands from **pHNbuffer** and push the commands into their submodules.

Five other modules - LA, RA, TR, LT, and RT - all have three degrees of freedom, and hence will look similar to the HN module, having three submodules.

C. *The Right Leg module*

As explained in the previous subsection, since the head has three degrees of freedom, the HN module had three submodules, each responsible for changes on an axis. However, the table-I shows that the TR (torso) module has only one degree of freedom, on the z-axis. Thus, the TR module will be much simpler as shown in the figure-7. Similarly, the three other modules LF, RF, and LL are also simpler modules with only one submodule for changing the angle γ.

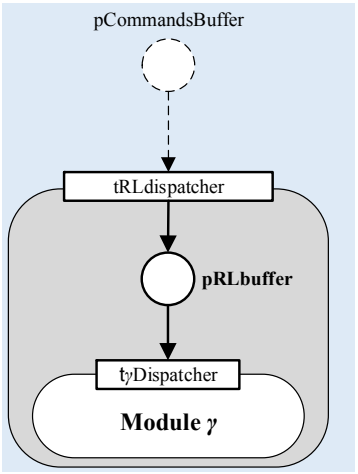


Figure 7. The Right Leg (RL) module.

V. THE SUBMODULES FOR CHANGING THE ANGLES

The submodules are responsible for realizing the commands to change the angles; as we have seen in the previous section, the HN module possesses three submodules for changing the angles on the x, y, and z-axes, whereas the RL module possesses only one submodule for changing the angle on the z-axis.

The figure-8 shows the alpha submodule for realizing the angular movement of Head & Neck along the x-axis. In this module, the transition **tαDispatcher** pulls the commands from **pHNbuffer**, the commands that are to change the angle α. After pulling a command, **tαDispatcher** also decode the command into a set of information:

- *Starting time*: if the value for the field 'start-at' is zero then **tαDispatcher** will fire immediately and

puts a token into **pα**, triggering the execution of the command. If the value for the field 'start-at' is not zero, then firing of **tαDispatcher** will wait until the clock has passed the time indicated by the value of 'start-at'.

- *Net angular displacement*: if the net angular displacement (explained in the *subsection-A* below) is positive then only **tα+ON** will be enabled thus passing the token into the cycle "**pα+** → **tα+** → **pα+**", which will increase the angle iteratively. Similarly, if the net angular displacement is negative then only **tα-ON** will be enabled thus passing the token into the other cycle "**pα-** → **tα-** → **pα-**", in order to decrease the angle iteratively.
- *Duration of action*: the value of the field 'duration' indicates the duration of the action, either for increasing or decreasing the angle to achieve its final value. If the angle is to be increased then it is the multiple firings of **tα+** (in the cycle "**pα+** → **tα+** → **pα+**") that increases the angle iteratively. However, after each firing of **tα+**, there must be a pause time, otherwise the net angular displacement will happen too quickly, due to the rapid firings of **tα+**. Calculation of the pause time is explained in the *subsection-C*.

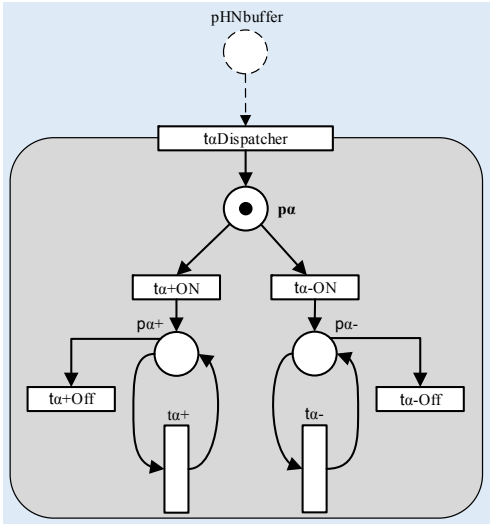


Figure 8. Module α for the angular movement of the head & neck (HN) on the *x* axis.

A. Calculating the Net Angular Displacement

A command demands change of an angle from its current value to its final value, via two actions: 'goto' or 'add'.

If the action is 'goto', then the final value of the angle will be the value of the field 'degrees'. Hence, *net angular displacement* ($\Delta\alpha$) will be the difference between the value of the field 'degrees' and its current value.

$$\Delta\alpha = (\text{value of 'degrees'} - \text{current value})$$

If the action is 'add', then the final value of the angle will be the value of the field 'degrees' added to its current value. Hence, net angular displacement ($\Delta\alpha$) becomes:

$$\Delta\alpha = \text{value of 'degrees'}$$

B. Calculating the Number of Firings

Taking α as the angle to be changed, let us assume that net angular displacement ($\Delta\alpha$) is a positive value. Hence, it is the multiple firings of **tα+** that will increase the angle.

Let us denote the angular increment per firing of **tα+** as $\delta_{t\alpha+}$. Let us also denote the number of firings of **tα+** that is needed to achieve the required $\Delta\alpha$ as $n_{t\alpha+}$. Hence,

$$n_{t\alpha+} = \Delta\alpha / \delta_{t\alpha+}$$

For example, in the fourth command shown in the figure-5, the angle α should be changed to -30 degrees in 'goto' action. Let us assume the following: the current value of α is 10 degrees, and the angular increment per firing of **tα-** ($\delta_{t\alpha-}$) is 0.5 degrees. Hence, $\Delta\alpha$ = -30 – 10 = -40 degrees. The number of firings of **tα-** needed to achieve this action $n_{t\alpha-}$ = 40 / 0.5 = 80. Thus eighty firings of **tα-** is needed to realize this action. After the eightieth firing of **tα-**, **tα-** will be disabled and **tα-OFF** will be enabled, thus the token in **pα-** that enabled the cycles of decreasing the angle iteratively will be taken away from the cycle by **tα-OFF**.

C. Calculating the Pause Time

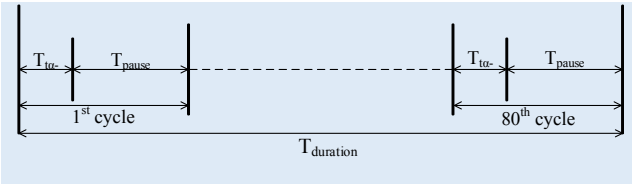
In order to explain what the pause time is, let us continue with the fourth command; this time, we will focus on the field 'duration'. We found in the previous subsection that to realize this command, the net angular displacement $\Delta\alpha$ becomes -40 degrees, and that we need eighty firings of **tα-**. In addition, the forth command states that the duration of the action ($T_{duration}$) must be 60 TU.

Let us assume that the firing time of **tα-** ($T_{t\alpha-}$) is 0.25 TU. If we allow **tα-** to fire its eighty firings one after the other, then the action of realizing the net angular displacement of -40 degrees will happen in 20 TU (80 X 0.25); this means, the action will complete much sooner in 20 TU than in 60 TU.

In order to make the action last over a period of $T_{duration}$, after every firing of **tα-**, it should be prevented to fire for a period of 'pause time'. The addition of pause time to firing time of **tα-**, makes the cycle time and that the eighty cycles should make up the required duration of 60 TU.

Cycle time: $T_{cycle} = T_{duration} / n_{t\alpha+} = 60/80$ *TU = 0.75 TU.*

Pause time: $T_{pause} = T_{cycle} - T_{t\alpha-} = 0.75 - 0.25 = 0.5$ *TU.*



Timing in executing the fourth command using multiple firings of **tα-**.

The figure-9 shows how the pause time makes the action lasts over the specified duration of time. The figure-10a shows the multiple firings of **tα-** without pausing after each firing; in this case, the action is complete in 20 TU, much sooner than the expected 60 TU. In the figure-10b, the firing cycles are padded with pause time, making the multiple

firings of **tα-** lasts over the period of specified duration, which is 60 TU.

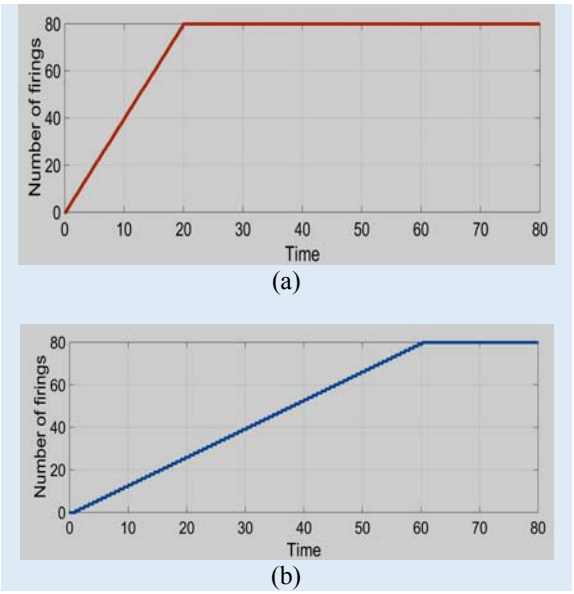


Figure 9. Realizing the duration of an action with pause time.

D. *The Submodules for the angles Beta and Gamma*

The submodules for the angles β and γ will be very similar to the submodule for α. The only difference is that, in the submodule for α, it is the transition **tα+** (or **tα-**) that fires to increase (or decrease by **tα-**) the angle α. In addition, it will be made sure that the resulting angle falls within the ROM of α. However, in the submodule for the angle β, it will be **tβ+** (or **tβ-**) that makes the angle β increase or decrease, and so forth for the submodule for γ.

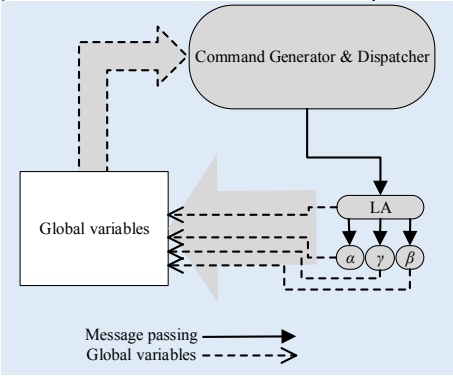


Figure 10. Closed-loop system, where communication is done thorugh message passing and global variables ('pigeon holes').

E. *The Commands Generator and Dispatcher module*

The proposed model is a closed-loop system, in which the commands are passed as messages from a higher order module to its submodules; information is passed from lower order submodules to higher order modules using global variables (or pigeon holes), as shown in the figure-11. For simplicity, only one module (LA) is shown in the figure-11.

VI. GPENSIM

GPenSIM is a toolbox of functions for MATLAB platform, offering a Petri Net language that realizes Petri Net theory along with and a set of utility functions. In GPenSIM applications, the basic model is a Petri Net model that is defined in a special Petri Net language. GPenSIM is being used by many universities in the world; recently, GPenSIM was claimed as the ideal tool for working with marked graphs [19]. GPenSIM - developed by the author of this paper - satisfies four criteria such as [17, 18]:

- Ease of use: anyone with limited mathematical and programming skills should be able to use the tool,
- Flexible: GPenSIM is applicable for modeling and simulation of discrete event systems in any domain (whether in engineering, business, or economics), as long as the system under scrutiny is discrete-event based.
- Extensible: the modeler can extend or replace any functionality available in the tool, and can also add newer functionality, and
- Compact simulation code.

In addition, GPenSIM also supports cost calculations and real-time control of discrete event systems.

A. *GPenSIM support for Petri net extensions*

Petri Net with a logical extension such as inhibitor arcs makes the other extensions (transition priorities, and enabling functions) redundant [19]. Though redundant, GPenSIM supports all the well-known logical extensions so that the modeler may choose the extension that he finds more comfortable to use. GPenSIM supports the color extension too, as this extension provides facilities to build a more compact and modular Petri Net models. GPenSIM also provides some special purpose extensions such as Cohesive Place-Transition nets with Inhibitor arcs (CPTI) [20].

B. *Activity-oriented Petri Nets*

Though Petri Nets could be used to model discrete event systems that involve a large number of resources, the resulting model could be huge due to the number of resources involved. Usually, even for a problem with few activities competing for a few resources, the resulting Petri Net model can be very large [14]. Activity-oriented Petri nets (AOPN) is an approach for obtaining compact Petri net models of discrete event systems where resource sharing and resource scheduling dominate [14]. GPenSIM supports AOPN too.

AOPN is a two-phase modeling approach: In the Phase-I, the static Petri Net model is created. In this phase, mainly the activities are considered, and they are represented by transitions embedded with places as buffers. The resources are grouped into two groups such as 'focal' resources and 'utility' resources. Only the focal resources are included in the static Petri Net model; the utility resources will be considered later in the phase-II (the run-time model). Thus, a compact Petri Net model is obtained with only the transitions representing the activities and, if there are any focal resources, they will be represented by places. Using the tool

GPenSIM, coding the static Petri net in phase-I will result in the Petri net definition file (DEF). In the phase-II, the run-time details that are not considered in the phase-I are added to the Petri Net model; e.g. transitions (activities) requesting, using, and releasing of the utility resources are coded in the run-time model. Using the tool GPenSIM, the run-time details in the phase-II will be coded in the files COMMON_PRE and COMMON_POST. The interested reader is referred to the GPenSIM user manual available from the website [15].

C. *Modular Petri Net model*

When developing larger Petri Net models, it is necessary to develop the model with modules. The newer version of GPenSIM allows modularization so that the flexibility of the models, as well as the comprehensibility, can be improved. In addition, modularization reduces the development time, and it increases the robustness. Modularization using GPenSIM is not a Petri Net extension; it is rather a modeling convenience.

D. *Industrial Applications of GPenSIM*

GPenSIM is being used to solve many industrial problems as well, e.g. in Aquaculture industry and Fish supply chain [21], airport capacity planning [22], production engineering [23], grid computing [24], e-commerce [25].

E. *The files for implementation using GPenSIM*

Modeling and implementation of a Petri Net model with GPenSIM involve *at least* four M-files (figure-12):

1. The Petri Net definition file (DEF): the static Petri Net graph is defined in this file. If the Petri Net model is composed of several modules, then each module can have their own DEF.
2. The main simulation file (MSF) is for assigning the initial dynamics and then for starting the simulation runs. Plotting and printing the simulation results are also coded in this file.
3. The common pre-processor (COMMON_PRE): this file is for coding the firing conditions that have to be satisfied before enabled transitions are allowed to start firing.
4. The common post-processor (COMMON_POST): this file is for coding the post actions, the actions that have to be executed when the transitions complete firing.

In addition to the common pre-processors, there can be specific pre- and post-files too, attached to specific transitions. Finally, the modules can have their own common pre- and post-processor files too.

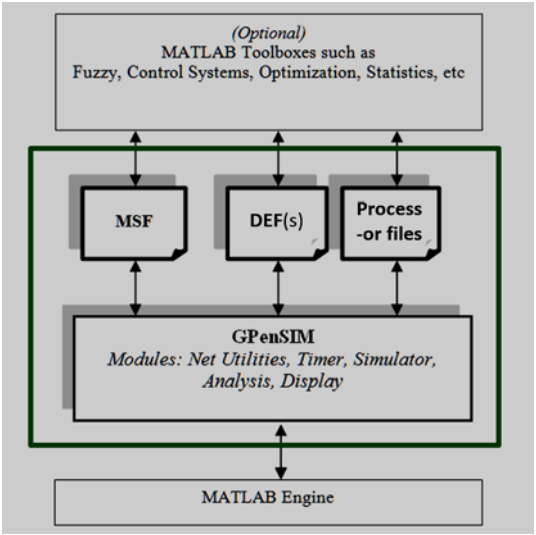


Figure 11. The files for implementing a Petri Net model in GPenSIM: MSF, DEF(s), and processor files (e.g. common pre- and post files).

VII. IMPLEMENTATION

For simulations, the Petri Net model is implemented on MATLAB platform using GPenSIM. The complete code is not given in this paper due to brevity; the interested reader is encouraged to inspect the code at the link provided as reference [26].

The following files are used for implementation of the Petri Net model for humanoid:

1. The main simulation file (MSF):
2. The Petri Net definition file (DEF) for the inter-modular connections: connector_def.m
3. The common pre-processor and the common post-processor 'COMMON_PRE.m' and 'COMMON_POST.m.'

In addition, there are 30 additional files: the DEF, the common-pre and the common-post processor files for each of the 10 modules and the three sub-modules.

The excerpts from the main simulation file (MSF) is given below; for simplicity, the MSF is for testing the Head & Neck (HN) module only; also, this module is tested with the alpha and beta sub-modules only.

```
% MSF FOR TESTING THE HEAD & NECK (HN) MODULE ONLY
% ONLY THE SUB-MODULES ALPHA AND BETA ARE TESTED

% GLOBAL VARIABLES & DATA
global global_info
global_info.STOP_AT = 440;

% LOAD THE ANGULAR MAXIMA; SET THE INITIAL DYN ARS
globalStaticConstants();
initDynamicVariables();

% DELCARE THE DEF
png = pnstruct({'HNa_pdf','HNb_pdf','test_HN_pdf'});

% DECLARE THE INITIAL DYNAMICS
startTokens = length(global_info.CENTRAL_COMMAND_Q);
% INTIAL TOKENS
dyn.m0 = {'pStart',startTokens};
% FIRING TIMES
dyn.ft={'tGlobalDispatcher',0.1,'tHNdispatcher',0.1, ...
        'tHNaDispatcher',0.1, 'allothers',0.1};
```

```
% INTIAL PRIORITIES OF THE TRANSITIONS
dyn.ip = {'tGlobalDispatcher',3,...
    'tHNdispatcher',2,'tHNaDispatcher',1};

% START THE SIMULATIONS
pni = initialdynamics(png,dyn);
sim = gpensim(pni);

% PRINT THE RESULTS
prnstate('Final state: ');
prnss(sim);
plotp(sim,{'pHNa-Off','pHNa+Off','pHNa','pHNa','pHNa+'});
```

A sample DEF file is given below. This DEF is for the module shown in the figure-8, the module α for the angular movement of the head & neck (HN) on the *x*-axis.

```
% DEF for the Head & Neck module
function [png] = HNa_pdf()
% NAME OF THE MODULE
png.PN_name = 'Head & Neck, Alpha angle on the X-axis';
% SET OF PLACES
png.set_of_Ps = ...
    {'pHNa+','pHNa+Off','pHNa+Counter',...
     'pHNa-','pHNa-Off','pHNa-Counter',...
     'pHNa', 'pHNa0'};
% SET OF TRANSITIONS
png.set_of_Ts = ...
    {'tHNa+', 'tHNa+ON', 'tHNa+Off',...
     'tHNa-', 'tHNa-ON', 'tHNa-Off',...
     'tHNa0','tHNaDispatcher', 'tHNaPause'};
% SET OF ARCS
png.set_of_As = ...
   {'pHNa+','tHNa+',1, 'tHNa+','pHNa+',1,
    'tHNa+','pHNa+Counter',1, ...
   'pHNa','tHNa+ON',1, 'tHNa+ON','pHNa+',1,...
   'pHNa+','tHNa+Off',1,'HNa+Off','pHNa+Off',1,...
   'pHNa-','tHNa-',1, 'tHNa-','pHNa-',1, ...
   'tHNa-','pHNa-Counter',1, ...
    'pHNa','tHNa-ON',1, 'tHNa-ON','pHNa-',1,...
    'pHNa-','tHNa-Off',1, ...
    'tHNa-Off','pHNa-Off',1,...
    'pHNa','tHNa0',1, 'tHNa0','pHNa0',1,...
    'tHNaDispatcher','pHNa', 1};
% SET OF INPUT/OUTPUT PORTS
png.set_of_IOPorts = {};
```

VIII. DISCUSSION AND CONCLUSION

Literature study reveals many works on modeling, simulation, and control of humanoid robots. Some of these works use Petri Nets as the main tool. These works have common attributes:

- Hierarchical architecture, where there are several layers (numbering two to four) of agents for control and coordination. The layers are to reduce the complexity due to the number of sensors and actuators involved and also due to a large number of discrete actions that are needed to make even a simple movement.
- The model is built with modular components; again, this is to reduce the complexity involved in building the model (usually large) and to make the model comprehensible and error-free.
- Petri Net model is used for verification of the model. For implementation, some works use the same Petri Net model for direct control, whereas the other works convert Petri Net into a high-level language like Java.

It is highly visible from the figures of the modules and submodules, that the design of the Petri Net model of a humanoid robot is modular. The modular design brings many advantages such as independent development, maintenance, and testing of the modules, easy to identify and place the functionality, reuse of code, etc. [27]; in summary, it is important to develop a modular model for increased reliability, maintainability, and portability of the system.

The communication infrastructure of the model uses message passing in the downward direction (modules to submodules) and global variables in the upward direction. This is because, the commands that are generated by the generator are numerous and are aimed for various modules; thus, a pipeline for directing the messages from top to bottom, and the modules pulling the messages aimed for them along the pipeline, seems a sensible strategy [28].

However, the current values of the angles, as there are only 22 of them (from table-I, six modules have three degrees of freedom, and four modules have only one degree of freedom, thus $6 \times 3 + 4 \times 1$), we can afford memory to keep these values as global variables make them visible to any interested agents (modules/submodules).

Further Work: The model as shown in the figure-11 is a closed-loop system; thus, the model can be used to control a humanoid, e.g. NAO robot, as shown in the figure-13. However, due to time limitations, testing the model-based control on a real humanoid like NAO is not yet complete. This work is still ongoing.

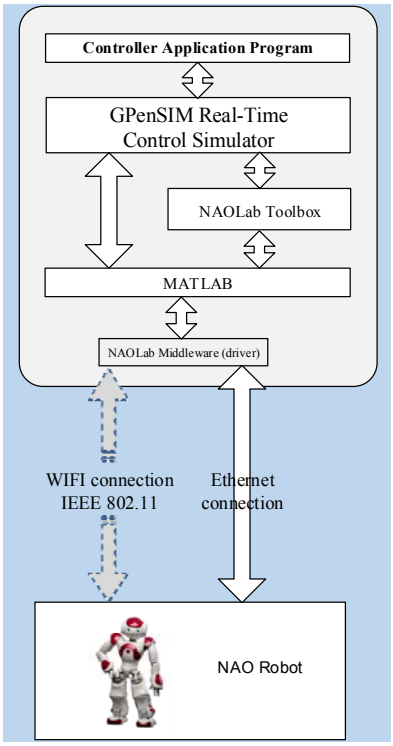


Figure 12. Discrete control of NAO robot with GPenSIM and NAOLab Toolbox.

REFERENCES

[1] B. Maharjan, Modeling Humanoid Swarm Robots with Petri Nets, MSc Thesis, University of Stavanger, June 2015.

[2] G. Fabrice, O. Bruneau, and F. B. Ouezdou. "Analytical and automatic modeling of digital humanoids," International Journal of Humanoid Robotics, vol. 2, No. 03, 2005, pp. 337-359.

[3] M. Arbulú, and C. Balaguer, "Real-Time Gait Planning for the Humanoid Robot Rh-1 Using the Local Axis Gait Algorithm," International Journal of Humanoid Robotics, vol. 6, No. 01, 2009, pp. 71-91.

[4] V. Potkonjak, S. Tzafestas, M. Vukobratovic, M. Milojevic, and M. Jovanovic, "Human-and-humanoid postures under external disturbances: Modeling, simulation, and robustness, Part 1: Modeling," Journal of Intelligent & Robotic Systems, vol. 63, No. 2, 2011, pp. 191-210.

[5] M. Panggabean, Modeling and simulating motions of human bodies in a futuristic distributed tele-immersive collaboration system for synthesizing transient input traffic, PhD Thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2014.

[6] K. Kobayashi, A. Nakatani, H. Takahashi, and T. Ushio, T. Motion planning for humanoid robots using timed petri net and modular state net. In Systems, Man and Cybernetics, 2002 IEEE International Conference on (Vol. 6, pp. 6-pp). IEEE, October 2002

[7] A. Lehmann, R. Mikut, and T. Asfour. "Petri nets for task supervision in humanoid Robots." VDI BERICHTE 1956 (2006): 71.

[8] R. Zollner, T. Asfour, and R. Dillmann. "Programming by demonstration: Dual-arm manipulation tasks for humanoid robots." Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. Vol. 1. IEEE, 2004.

[9] D. Osswald, N. Gorges, and H. Worn. "Reactive hand-arm coordination for a humanoid robot using extended petri-nets." Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on. IEEE, 2007.

[10] J. S. Lee, and P. L. Hsu. "Remote supervisory control of the human-in-the-loop system by using Petri nets and Java". IEEE Transactions on industrial electronics, 50(3), 431-439, 2003.

[11] A. Faller, M. Schunke, and G. Schunke, The Human Body: An Introduction to Structure and Function, Georg Thieme Verlag, 2004.

[12] National Aeronautics and Space Administration (NASA), Anthropometry and biomechanics (Vol. 1, Section 3) in Man-Systems Integration Standards (NASA-STD-3000) Revision B, 1995.

[13] G. Ciardo, "Toward a Definition of Modeling Power for Stochastic Petri Net Models," Proc. International Workshop on Petri Nets and Performance Models, 1987, pp. 54-62

[14] R. Davidrajuh, "Activity-Oriented Petri Net for scheduling of resources." Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on. IEEE, 2012.

[15] GPenSIM: Accessed on 20 January 2017: http://davidrajuh.net/gpensim

[16] A. Cameron, M. Stumptner, N. Nandagopal, W. Mayer, and T. Mansell, "Rule-based peer-to-peer framework for decentralised real-time service oriented architectures," Science of Computer Programming, 97, 2015, pp. 202-234

[17] R. Davidrajuh, "Developing a new Petri net tool for simulation of discrete event systems.", In Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on (pp. 861-866). IEEE., 2008.

[18] R. Davidrajuh, "Developing a Petri Nets based Real-Time Control Simulator," International Journal of Simulation, Systems, Science & Technology (IJSSST), vol. 12, issue. 3, pp. 28-38, November 2012

[19] G. Ciardo, "Toward a Definition of Modeling Power for Stochastic Petri Net Models," Proc. International Workshop on Petri Nets and Performance Models, 1987, pp. 54-62

[20] R. Davidrajuh and N. Saddallah, "Implementation of "Cohesive Place-Transition Nets with Inhibitor Arcs" in GPenSIM". IEEE 2016 Asia Multi Conference on Modelling and Simulation, Kota Kinabalue, Malaysia, December 4-6, 2016.

[21] R. Melberg and R. Davidrajuh, R. "Modeling Atlantic Salmon Fish Farming Industry: Harvesting Sub Model." World Academy of Science, Engineering and Technology, Vol. 57, 2009, pp. 1086-1096, 2009.

[22] R. Davidrajuh and B. Lin, "Exploring airport traffic capability using Petri net based model," Expert Systems with Applications, 38(9), pp. 10923-10931, 2011.

[23] B. Skolud, D. Krenczyk, and R. Davidrajuh, "Solving repetitive production planning problems: An approach based on Activity-oriented Petri nets." 11th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO), San Sebastian, Spain, October, 19–21, 2016.

[24] A. Azab, H. Meling, and R. Davidrajuh, "A Fuzzy-Logic Based Coordinated Scheduling Technique for Inter-Grid Architectures," K. Magoutis and P. Pietzuch (Eds.): DAIS 2014, LNCS 8460, pp. 171–185, 2014.

[25] R. Davidrajuh, "Distributed Workflow based Approach for Eliminating Redundancy in Virtual Enterprising," Journal of Supercomputing (ISSN: 0920-8542), 63(1), pp. 107-125, 2013.

[26] Code for simulation: Available from 20 October 2015: http://davidrajuh.net/CONF/2015-AIMS-hr.htm

[27] M. O'Neil, Modular Advantage: Six Benefits of a "Pay as You Go" Approach to Data Centre Design, Canadian Market Insight from IT Market Dynamics, August 2011.

[28] W. Gropp, Tutorial on MPI: The Message-Passing Interface. Mathematics and Computer Science Division Argonne National Laboratory Argonne, IL, USA, 2009.