University of Stavanger

**Faculty of Science and Technology**

# MASTER'S THESIS

| | |
|---|---|
| Study program/ Specialization:<br><br>Master of Science in Computer Science | Spring semester, 2015<br><br>Open / Restricted access |
| Writer:<br><br>Bikee Maharjan | …………………………………………<br>(Writer's signature) |

Faculty supervisor:

Reggie Davidrajuh

Thesis title:

**Modeling Humanoid Swarm Robots with Petri Nets**

Credits (ECTS): 30 ECTS

| | |
|---|---|
| Key words:<br><br>Humanoid Robot, Swarm robots, Petri Net , GPenSIM, Forward Kinematics, Gait Cycle ,Range of Motion | Pages: 97<br><br>+ enclosure: CD<br><br>Stavanger, 29th June, 2015 |

# Modeling Humanoid Swarm Robots with Petri Nets

Bikee Maharjan

Faculty of Science and Technology
University of Stavanger

June 2015

# Abstract

Robots have become a hot topic in today's electronic world. There are many definitions for it. One of the definition in Oxford dictionary states "a robot is a machine capable for carrying out a complex series of action automatically especially one programmable by a computer". This paper deals with a special kind of robot, which is also known as humanoid robot. These robots are replication of human beings with head, torso, arms and legs.

A model of human is presented in this paper as discrete event system adapted from "Modeling and simulating motions of human bodies…"[1]. This model consists of sixteen interrelated limbs defined in 3D space, so most limbs/joints are able to make movement in three different angles (**α**, **β** and **γ**). Full details regarding Range of Motion (ROM) of rigid body in forward kinematic is illustrated. Human motions are categorized into two types: stochastic and deterministic motions. Deterministic motions are demonstrated using gait cycle of walking and running of normal adult person.

The main focus of this paper is to model and simulate humanoid robot represented as Discrete Event Systems (DES); in Petri Net using GPenSIM and later expand those group of robots to swarm setting. GPenSIM is General Purpose Petri Net simulator [2] developed as toolbox for MATLAB to model and simulated discrete events using Petri net tools. Each joint's angle is treated as a separate Petri Net model which is independent from each other and their movement's limits are defined by ROM of normal human body. The instructions relating to the motion of joints for simulation are fed through a file to the instructor. These movements of joints are represented by variation of tokens displayed at the end of simulation in a graphical figure. Further, same structure of model is used in swarm of robots. Instead of feeding instructions to individual robots, a central instructor is created. This instructor acts as a master to robots acting as slaves where slaves include some predetermined commands embedded inside them. With central command system, a proper synchronization is achieved among group of robots working as swarm. A normal routine of group dance or simple group sport can be accomplished with calculated instructions on this swarm of robot.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Reggie Davidrajuh for his continues support with vast knowledge, motivation and valuable guidance. His patience and support helped me all the way from beginning to end. I couldn’t have asked for any better advisor and mentor for this work.

I would also like to take this opportunity to thank all lecturers in the department of science and technology at University of Stavanger for their valuable knowledge and discussions which further helped me to improve my understanding in this field.

Most importantly, this would never have been possible without love and support of my parents who always have been there for me. Their constant encouragement and inspirational words always helped me to overcome setbacks and keep my focus on my study.

My sense of gratitude also goes out to all friends in Nepal, Norway and abroad who directly or, indirectly have provided their valuable support to achieve this goal.

Bikee Maharjan
University of Stavanger

# Table of Contents

# List of Figures

# List of Tables

Chapter 1

# 1 INTRODUCTION

Today robots have become a well know term. With all this rapid developments in technology, robots have been helping human lives to become easier every day. When we think of robots as ones that imitates like living beings mostly humans as shown in several movies like 'terminator', ' iron man', 'I, robot' and so on. However currently those type of robots only resides in our imagination and sci-fi movies because still our technology hasn't reached to that stage where an artificial intelligence includes common sense along with ability to interact with our everyday changing world.

Currently, we see robots obey commands and work in fields which are very dangerous, boring, repetitive or onerous [3] for humans. There are millions such robots being deployed all around the world in the sector of medical, space and mostly production industries.

There is no any exact definition of robot but there are some essential characteristics like sensing, movement, energy and/or intelligence that separate a robot from any normal machines. As defined by robotics [3]"robot is a system that contains sensors, control systems, manipulators, power supplies and software all working together to perform a task". There are lots of stuffs involved like the blending of physics, mathematics, computing along with mechanical and structural engineering for designing, building, programming and testing a full working robot.

When we talk about humanoid robots, they are the robots which are replication of humans as their name defines. They have torso, arms, legs and a head with same body structure of a human. Basically their appearance separates them from other robots. Some more advanced robot can even mimic human facial features. The main reason behind the construction of humanoid robots is for better understanding of humans by simulating them. Furthermore these robots are also used to perform various tasks for humans like personal assistance, as worker in manufacturing line and now days as a means of entertainment.

Swarm robotic basically deals with design and study of group of robots which are either controlled by any central instructor or robots coordinating with each other without depending upon external infrastructures [4]. In terms of swarm humanoid robots, the robots works in a distributed robotic system [5] which are heterogeneous, dynamically connected and autonomous system that performs its task in three dimensional environments.

A humanoid robot looks similar to humans, with a body shape built with a torso, a head, two arms and two legs. This project is dedicated to look for possibility of coordinating a group of (swarm) humanoid robots, performing a robotic group performance such as a sports or a dance. In situation like football match where coordination between each individual robot is compulsory.

This project is staring point towards group coordination and control of swarm humanoid robot. ***The objective of this project is to model a single humanoid robot in Petri Net and later expand the model to simulate in a swarm setting for multiple (group) coordination using Petri Net***. It is really complex task to create a realistic and correct simulation of human motion under full restriction of human movement.

Petri net is one of the powerful modeling and simulation tool since humanoid robots considered for this work performs discrete events which are almost dynamic and concurrent [6]. It has tools to represent complex mathematical and analytical interactions [7]. Even after being developed over half century ago solely to design automata models [8], it is now being adapted to create application to model communication protocols [9], manufacturing systems [10], cyber-intrusion [11], road traffic [12], games [13, 14], track driven traffic [15], robot movements [16] , modeling and controlling of discrete event system [17] and many more. Petri Net is also considered to be starting point for machine learning systems [18]. One of the main reason to consider Petri Net over other Discrete Event modeling tool is that it is difficult to lose visual information in Petri Net since whole process can be easily monitored [19] with flow of tokens.

GPenSIM is used in this project to implement humanoid robot using Petri Net tools. GPenSIM is a general purpose Petri Net simulation tool which is written in MATLAB language. The advantage of being developed in MATLAB is that integration with other useful tools in MATLAB is seamless and structure of code and syntax are easy to understand. The humanoid robot considered for this project is viewed as collection of discrete event systems. The flexibility, extensibility and ease [2] of use of GPenSIM to model and simulate such systems provides an edge over other such Petri Net simulation tools.

## 1.1 Adaptation

Modeling a humanoid robot is a very complicated process. There have been many researches in this field for years. The exact system envisioned to model a humanoid doesn’t exist till date. But, nevertheless with all the rapid advances in development of systems with immense complexity, makes way for analytical understanding which gives an edge for design and simulation. In this project, DES (Discrete Event System) simulation is considered; as it is a very powerful tool that can emulate complex dynamics of discrete events systems [20]. This project deals with simulation of movement of different human part via forward kinematic with uniquely reproducible motions. This simulation includes both stochastic and deterministic motions.

This project adopts the structure for design of the humanoid robot from the paper “Modeling and simulating motions of human bodies in a futuristic distributed tele-immersive collaboration system for synthesizing transient input traffic”. This paper describes details of a human body presented as a discrete-event system alongside ROM of stochastic human motion which is based on the forward kinematics of rigid human body in 3D space. Also deterministic motions of normal human are presented with gait cycles of running and walking and stochastic motions with ROM of normal humans. All basic requirements to represent normal human as discrete event in this project are based on this paper.

## 1.2 Organization of the Thesis

This thesis is organized in following way

- *Chapter 2* presents fundamental background necessary to understand the work done in this thesis.
- *Chapter 3* revolves around modeling of normal human body using Range of motion of healthy adult person as Discrete Events System.
- *Chapter 4* provides implementation details of modeling and simulating those discrete events as humanoid robot in Petri Net with GPenSIM.
- *Chapter 5* presents various test results; with different instructions in single and multi robot mode.
- *Chapter 6* sums up the thesis with some discussion and possibility of its implementation in real physical robot.

Chapter 2

# 2 BACKGROUND

## 2.1 Modeling

Modeling is a method of creating a model. A model simply can be termed as a representation of the structure and working mechanism of a system [21]. It is created to better understand the system by looking at it in a very simple way. This gives an insight look on the effects with various changes. Thought models are simpler version; it shouldn't be very much off from the real system and should include the entire major feature that defines that particular system. It is really difficult to balance a model between simplicity and realism. So, a good model is judicious tradeoff between them. A perfect model should be ones which are easier to understand and experiment with not losing the feature of realism. Model validation is very well known techniques where the model is simulated under known inputs and later those output obtained are compared with system's output. This determines how close the model has been compared to the real system. Mostly, these created models used as mathematical model for simulation studies.

The assumption made to create a model can be a mathematical expression, logical and symbolic expression which defines the relation between the objects of the system. After development these models can answer the question "what if" for system in real world.

## 2.2 Simulation

A simulation is [22] replication of operation of the model of a system or real world process. Since experimenting with the real system might not be practical or might be too expensive, simulation is an effective way to reconfigure and experiment in many ways with the model which doesn't seem to be possible with the real system. With the help of simulation, prediction can be made for potential changes that could affect the performance of the system. Simulation is also very useful during design phase of the system. Those experiments done during design phase lead to creating a robust system. Therefore, simulation can be used as a tool for evaluation of a real system for its performance as well as a tool to design which can foresee the performance of the system in various changes that could take place in the real world system. Bottleneck and over utilization of resources can be spotted during simulation for the better system performance.

In simple models used for evaluation of system there are only 1 or 2 numerical parameters. These are easier to compute using different mathematical techniques like calculus, probability theory or others. The real world systems are normally very big and complex. So, mathematic evaluations of their models become complicated and somewhat impossible. In these scenario, computer based simulations comes in handy. The behavior of the model are observed over time and data generated are collected which are later processed to measure the performance of the system. Historical data are also important for feeding the simulator for a realistic analysis. But the problems with such data are that there should be consistency and nothing should be changed either in operation or used material through the whole system process.

Most of the time simulation contributes a lot in creating a robust system or/and analysis performance of system. So, simulation can be used when [23]:

- Performing experiments on the real system become too expensive, infeasible and/or disorderly.
- Directly applying mathematical and /or analytic methods won't work
- Measurement of performance has to be done on a system over some period of time
- Comparisons has to be made between different designs and operating policies on the same system and figure out the best ones that fulfill the required specifications.

There are different simulation model differentiated by their characteristics which are described below:

- **Static or Dynamic**

  Simply a static simulation represents a state of a system at a particular time where as in dynamic model; the interactions of variables take place in different instance of time, so time plays a major role.

- **Deterministic or Stochastic**

  In deterministic, there are no involvements of variability of the input and output parameters so basically no random variable are in use. So, same set of inputs always result in same set of outputs.

  Unlike deterministic, stochastic contains numbers of random variables which defines the overall process of the system. The outputs of the simulation are random hence a true characteristic of system can only be estimated through varies number of runs to determine the expected performance of the model.

- **Continuous or Discrete**

  A continuous simulation is the one in which the state variables change continuously. For example cars travelling in a highway, flow of oil in a pipe, flight simulators; in cases like this the state variables changes continuously with respect to and/or of their position and velocity.

  In discrete, the changes in the state variables take place at a countable (finite) time in any range of given time. An example can be a simulation of restaurant, here state variable i.e. number of customers which increase when they arrive and decrease when they had their food and left. So, at discrete point in time, number of customers' changes at finite rate.

  Even though discrete simulation is less detailed in comparison to continuous, it is widely used for its simplicity to implement for variety of situations.

In this project we are focusing on the discrete-event simulations. As described in discrete simulations, modeling of the system are done in which the variable state values change at a discrete set of points in time. Here rather than analyzing models by analytical method, numerical methods are put in perception. Simulation models are "run" rather than "solved" in numerical method. This creates an artificial history of the system based on the assumptions. The observation results are later analyzed and hence system performance measures are estimated. The simulation models for the real world systems are normally very huge so, the amount of data needed to store and manipulate are beyond the capabilities of a human being. To cope with these problems, simulation runs are computed with the help of a computer. A simple flow chart shown in Figure 2-1 from [22] shows the steps in simulation study and is briefly described below:

- *Problem Formulation* : There should be a problem and that should be clearly understood by the description provided by the analyst and all policymaker should come to an agreement with the formulation
- *Setting of objectives and overall project plan*: A vision must be there for an objective to be achieved by the simulation. This should also indicate which simulator would be used and what type of methodology is appropriate for the formulation of problem.
- *Model Conceptualization*: The main art of conceptualization is the ability to figure out the essential features of a problem, make changes to basic assumptions and break the model down until a better approximation is found. So, start with simple model and then slowly move on to more complex.
- *Data collection*: As the model's complexity changes, the requirement of the data also changes along it. Data collection takes lot of time to perform simulation so, it better to start from beginning when model building is less complex.

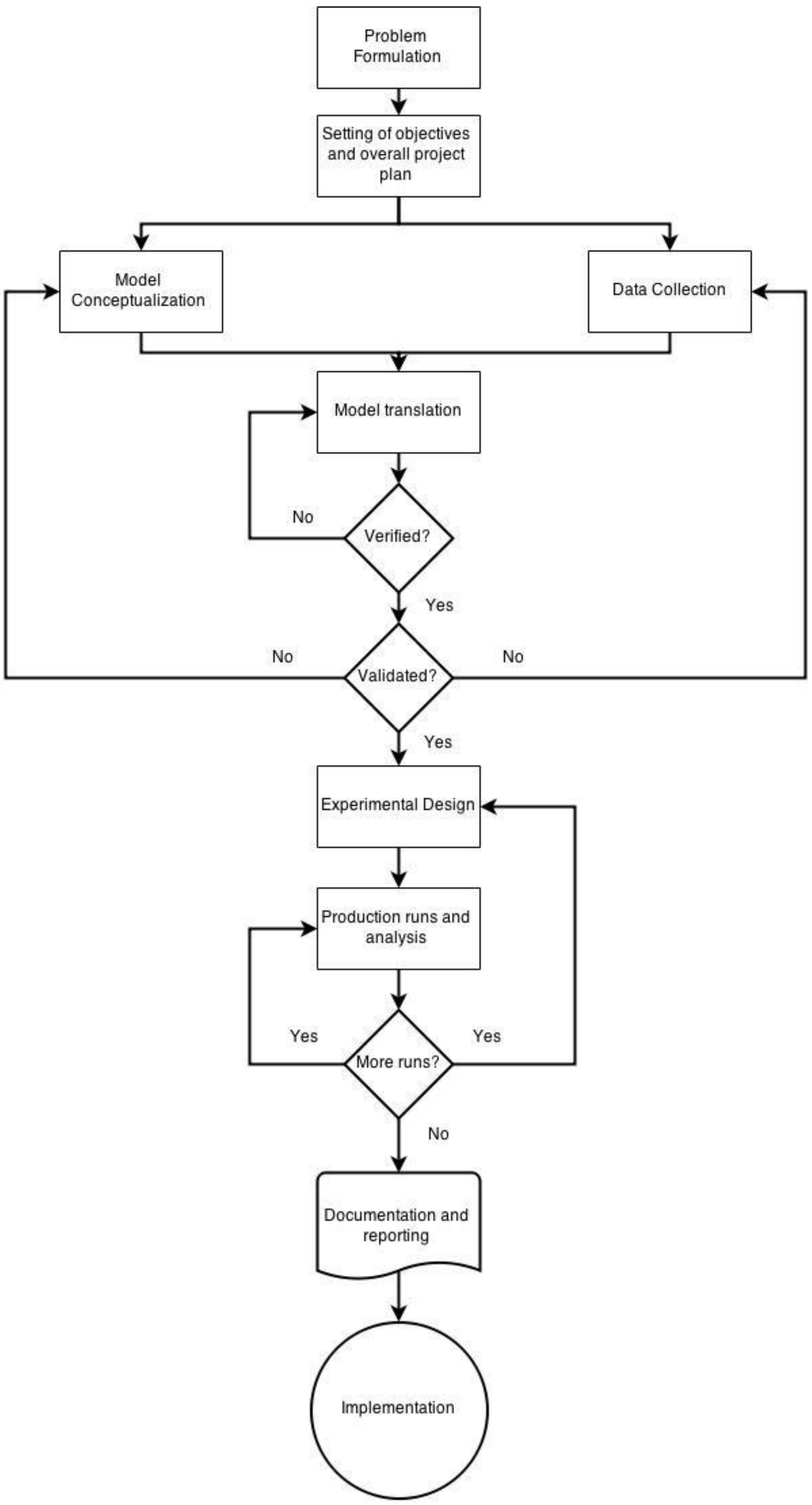


**Figure 2-1: Flowchart representing steps of simulation study [22]**

- *Model translation*: After conceptualization and data collection, models should be entered into computer in the format it recognizes since real world systems require great amount of information to store and compute and it's near to impossible to solve it manually.
- *Verified*: Verification must be done if the program created is ready for the simulation of complex build models in a proper manner. It definitely takes lots of debugging and fixing of logical structure along with input parameter for complex models.
- *Validated*: The created simulation model must accurately represent the real system which is done by calibrating the model by iteratively comparing and adjusting the model with actual one.
- *Experimental Design*: To determine if the simulation is working properly, there should be some alternatives to simulate functions which are already run and results are analyzed.
- *Production runs and analysis*: Analysis of the production run takes place to estimate measures of performance of system of the simulation being taken into account.
- *More runs*: The requirement of more runs is determined based on the runs that are completed and analyzed and also suggests additional experiments if required to follow.
- *Documentation and reporting*: Whole process should be documented in a fashion that if it is to be used in future, other analyst must be able to understand both the program and progress.
- *Implementation*: This phase's success totally relies upon how well all above steps have been executed. And also how close and deep the analyst got ultimate model involved during the whole simulation process; has its parts for the successful implementation.

## 2.3 Petri Net

Petri Net (PN) is graphical and mathematical (analytical) modeling tool [24] used mostly to model and simulate discrete events of distributed asynchronous, parallel, nondeterministic and concurrent systems. This concept was submitted by Carl Adam Petri in 1962 at Bonn University in Germany as his PH.D. thesis work [25]. These are basically special class for generalized graphs or nets. PNs are graphical tools which can be used to represent visual-communications like flow-charts, block diagrams and networks. The best part of PN is that it can be used by both theoreticians and practitioners. This can also work as powerful communication bridge between them. In one side practitioner can learn extra about making models more methodical where as on the other hand theoreticians get insight knowledge on creating more realistic models from the practitioner.

Later, PN was also proposed for designing various applications because of its generality and permissiveness. Any system or area which can be described graphically like flow charts and needs some concurrent or parallel activities can make use of Petri nets. But it comes with a price i.e. tradeoff between analysis capacity and modeling generality [24]. So, more simpler/general the model, difficult to perform analytical analysis. Petri Nets have a greater weakness over complexity problems. Without making some special changes or some restrictions depending upon the application, it is very complicated to analyze even modest sized system because it gets way too big. With some changes, there are many promising areas of its use like concurrent and parallel programs, discrete event systems, fault- tolerant system and so on. It is also very successful in applications to analyze performance of a system and design communication protocols.

A Petri net can also be termed as a directed graph having an initial state [24]. It is also weighted and bipartite graph consisting of two elements: places and transitions. Arcs connect places with transition and vice versa. While representing Petri net in graph; places are represented as circles and transition as boxes or bars. The arcs are weighted (positive integers) and a k-weighted arc corresponds to set of k parallel arcs. It is not necessary to label if the weight of arc is one. The marking assigned to place, transition or arcs can't be negative integer. The assigned marking to the place is denoted by tokens. If there are k nonnegative integers assigned to place p then the place is marked with k tokens. These tokens are represented graphically by black dot/s inside the circle which represents places. Tokens relates to objects that can flow between different nodes in a network. If tokens are too large to be placed graphically then the number of tokens can be shown with numbers rather than black dots. One example is shown below in Figure 2-2.

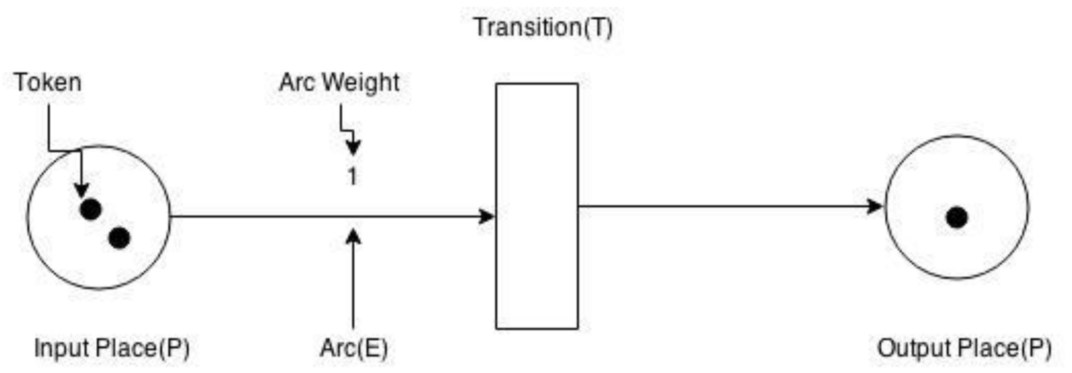


**Figure 2-2: Graphical representation of a Sample Petri Net Model**

During modeling of Petri Net, places are identified as conditions and transitions as events .It follows the concept of conditions and events. The transition has certain number of input and output places which represents the pre and post conditions corresponding to the event. The tokens in those places indicate truth of the condition relating to the place that can be explained as total number of resources available. Some of these interpretations of transitions with their input and output are shown below in Table 2-1.

| **Input Places** | **Transitions** | **Output Places** |
|---|---|---|
| Preconditions | Events | PostConditions |
| Input Data | Computation step | Output data |
| Input signals | Signal processor | Output signals |
| Resources needed | Task or job | Resources released |
| Conditions | Clause in logic | Conclusion(s) |
| Buffers | Processor | Buffers |

**Table 2-1: Some Typical Interpretations of Transitions and Places [24]**

Formal definition of classic Petri nets according

A Petri Net is a 5-tuple [24], $N = (P, T, F, W, M_0)$ where:

$P$ is a finite set of places, $P = \{p_1, p_2, . . . , p_m \}$,

$T$ is a finite set of transitions, $T = \{t_1, t_2 , . . . , t_m\}$,

$F$ is a set of arcs (flow reaction) from place to transition and vice versa,

$F \subseteq (P \times T) \cup (T \times P)$,

$W$ is a weight function, $W:F \rightarrow \{1,2,3,....\}$ and

$M_0$ is the initial marking, $M_0 : P \rightarrow \{0,1,2,3,...\}$.

Where $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$ .

A Petri net structure without any specific initial marking is denoted by *N*, *N = (P, T, F, W)*. So, with an initial marking the Petri Net is denoted by *(N, $M_0$)*.

Petri Net structure can be denoted by *N = (P, T, I, O) [26]*, where I is the transition input relation and O is the transition output relation.

The system states and changes are used to express behaviors of many systems. Here are transition (firing) rules used to simulate the dynamic behavior of a system, a state or change in marking of Petri nets. [24]

- A transition t is said to be enabled if each input place *p* of *t* is marked with at least *w (p, t)* tokens, where *w (p, t)* is the weight of the arc from *p* to *t*.
- An enabled transition may or may not fire (depending on whether or not the event actually takes place).
- A firing of an enabled transition *t* removes *w(p, t)* tokens from each input place *p* of *t* and adds *w(t, p)* tokens to each output place *p* of *t*, where *w (t, p)* is the weight of the arc from *t* to *p*.

It is possible for transition to not have input or output places. The ones without input are called source whereas without output are called sink. For a Petri Net to be pure it should not have self loops (i.e. place *p* can't be both input and output to a transition *t*). All ordinary Petri Nets have arc weights equal to 1.

The above described rule is illustrated in an example with Figure 2-3 and Figure 2-4. For a bicycle manufacturing, it requires 2 wheels and 1 body frame. Here the event of putting wheels and frame together is represented by transition *t*. The wheels are tokens present in the place *p1* and frames are tokens present in *p2*. *p3* is place for holding place for final product. The arcs correspond to the connection between events and conditions. Describing in basic Petri Net terms:

- Two input place p1(wheels) and p2(frame)
- One output place p3(bicycle)
- Transition t(event of combining)
- Arc from p1 to t weights 2 (2 wheel requires for one bicycle)
- Arc from p2 to t weights 1(1 frame for one bicycle)
- Arc from t to p3 weights 1(one bicycle manufactured)
- Two tokens in p1 (2 wheels available)
- Two token in p2(1 frame available)

According to the rule, for the transition *t* to be enabled, both input places must have number of token equal or greater than the arcs weight that connect them to the transition. Since both the input places have 2 tokens, transition *t* is fired. After firing transition *t*, initial marking (2,2,0) stating *p1*, *p2* and *p3* has 2 ,2 and 0 token respectively will be changed to new marking (0,1,1) which is shown in the figure 6. There are no enough tokens available in p1; hence transition t is no longer enabled.

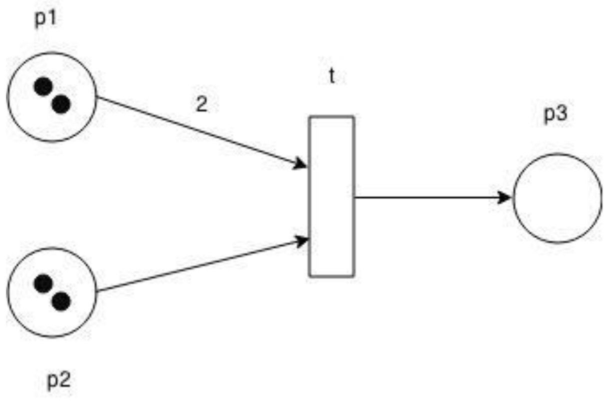


**Figure 2-3: The marking before firing the enabled transition t**

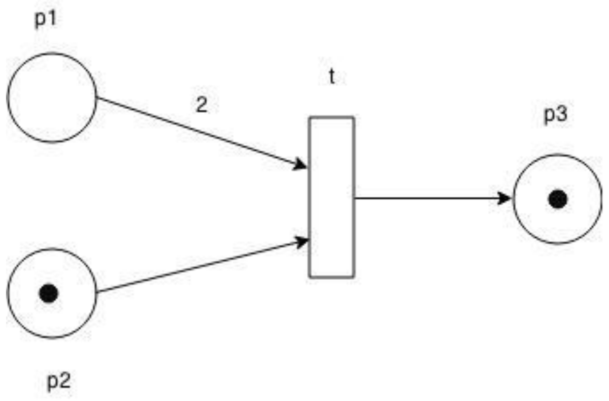


**Figure 2-4: The marking after firing t, where t is disabled**

## 2.4 Extended Petri Nets

The ordinary Petri net discussed is not capable for modeling a real world system as there are so many parameters involved. Also, ordinary PNs are not able to model Turing machine [27]. There are many extensions to Petri Net which increases the modeling power of PNs. Everything comes with a price. Extensions of Petri Nets decrease the analytical capacity of the PNs. But various subclasses of Petri Net are put forward to increase the analytical power which is done mostly be restricting the structure of the Petri Net. Some extensions are backward compatibility with original PNs and some add properties to the original PNs. Those extensions that are possible to be modeled in original Petri Nets can't be termed as extended Petri nets. Some of the popular extensions are discussed below:

### 2.4.1 Timed Petri Net

In real world systems, time plays great role. In ordinary Petri Net there is no mechanism of dealing with time which makes it difficult to model real world working system. Transitions take zero time to fire in classical Petri Net which is called "primitive transitions" but in case of Timed Petri net, transitions take deterministic or stochastic time (i.e. "non- primitive transitions").
This is defined in [28] as

A Timed Petri Net is a 6-tuple TPN = (P, T, A, $m_0$, D) where:

- PN(TPN) = (P, T,A, $m_0$) is a classical Petri Net and
- D: $t \rightarrow R^+$ is the duration function, a mapping of each transition into a positive rational number, meaning firing of each transition t, now takes exactly $dt_i$ time units.

### 2.4.2 Colored Petri Net

This extended Petri Net is like addition of extra feature to the classical Petri Net. It was developed in 1970 [29]. In Petri net, tokens are used to represent objects like resources, humans etc. Colored Petri Net was developed in the concept of adding attributed to those tokens/objects. Here, every token has a value referred to as 'color'. These values are manipulated in transitions. Transitions can then use those values of consumed tokens and determine the value of produced token. With values assigned to tokens as colors, it is also possible to add 'preconditions' to check for the color of token to be consumed.

### 2.4.3 Inhibiting arc

Inhibiting arc is the simplest extension to Petri Net that allows zero testing. It is an arc similar to normal arc but has a circle at the end instead of an arrow [27] .The concept of circle is borrowed from switching theory where a small circle is seen as "not". This arc is only from place to transition. With addition of this arc changes the firing rule as follows:

- A transition is enabled when there are enough token in input (normal) place and zero tokens in all its inhibitor inputs.
- After firing, the transition removes tokens from only normal input places.

General case:

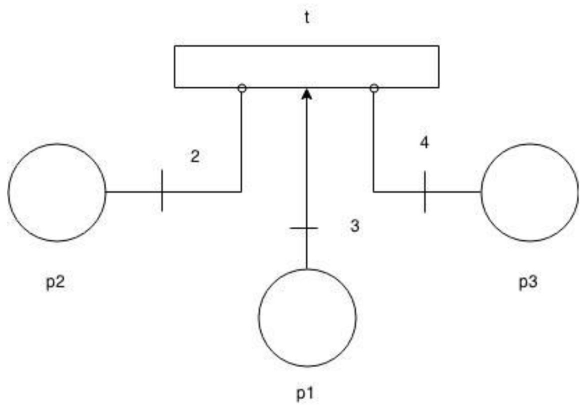


**Figure 2-5: Inhibiting Arc**

**T is enabled if:**

p1 >= 3,

p2 < 2 and

p3 < 4

Similarly, **T is disabled if:**

p1 < 3 ,

p2 >= 2 or,

p3 >= 4

## 2.5 GPenSIM

General-Purpose Petri Net Simulator (GPenSIM) was developed by Reggie Davidrajuh [2]. It is one of the advanced tools used to model complex and large systems because of it flexibility, extensibility and ease of use. GPenSIM in simple terms can be described as toolbox built for/in MATLAB platform. The MATLAB environment has lots of diverse toolboxes like Control Systems, Fuzzy Logic and many more. These can be combined with GPenSIM. In simple Petri Net model all the details are in frontline but GPenSIM pushes its resources to background so it becomes more flexible to model and simulate. For more GPenSIM also supports extended Petri Net like Colored Petri Net, Inhibiting arc, stochastic timing, transition priority, resource management along with tools like coverability tree, internal clock and many more for better modeling and analyzing the performance of Petri Net of a real system.

In this thesis we are dealing with simulation of discrete event systems. There are some famous simulators currently in use and they are Automata, State flow and Petri nets. Amongst them Petri net is widely used for modeling and simulation as it is very easy to learn and use. These tools are mostly available free of charge and are capable of modeling large complex systems. It is very difficult to integrate these Petri Net tools written in high level languages like Java or C++ with other toolboxes as they operate independently. So, with GPenSIM being written in MATLAB, integration becomes way too simple as MATLAB houses many useful toolboxes.

### 2.5.1 Architecture of GPenSIM

GPenSIM was build under the well proven paradigms in software engineering which includes layered architecture, modular components and natural language interface [2].

#### *2.5.1.1 Layered architecture*

As seen in the Figure 2-6, GPenSIM is based upon 3-layer architecture design namely: Liner Algebraic Layer, Presentation Layer and Application Layer. The bottom layer / Liner Algebraic Layer handles the run time dynamic of the Petri Net which is done by using linear algebraic equations and matrix manipulations for generation newer states. The middle /Presentation layer deals with appending more user interactive functionality like stochastic timing, coloring of tokens, user-defined conditions and much more. Finally, the top/application layer is application which are used by user to interact by constructing Petri Net based models, simulating it , figuring coverability tree, generation of output results as text or graphs and many more.

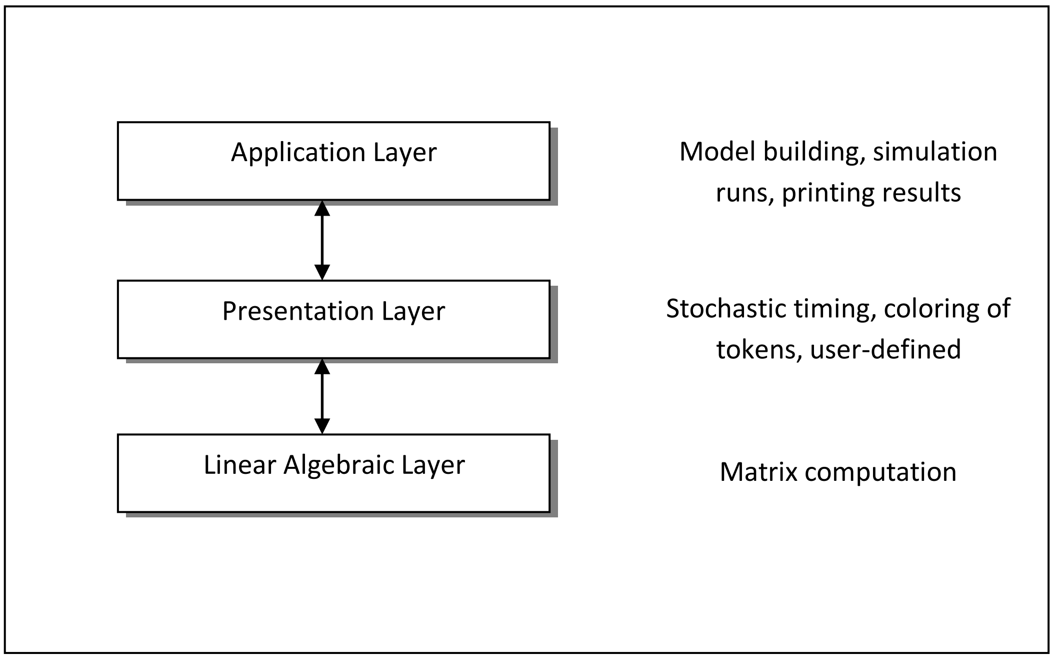

Application Layer
Model building, simulation runs, printing results
Presentation Layer
Stochastic timing, coloring of tokens, user-defined
Linear Algebraic Layer
Matrix computation

**Figure 2-6: 3- layer architecture as described in [2]**

#### *2.5.1.2 Modular Components*

As a modeling and simulation tool for discrete event system in MATLAB, GPenSIM consist of some modules in the form of files. The main simulation file (MSF) and Petri Net Definition file/s (PDFs) are the main operating files for GPenSIM. There may be various numbers of PDFs depending upon number of models used. In additions to these files, there can also be Transition Definition Files (TDFs) i.e. pre-processors and post-processors. These may be in multiple numbers depending upon the requirements of pre and post conditions to a transition in Petri net model. GPenSIM has a packet which can be used to exchange values between different modules called '*global_info*'. Since all files can access the packet, any file can read and even modify the value inside it.

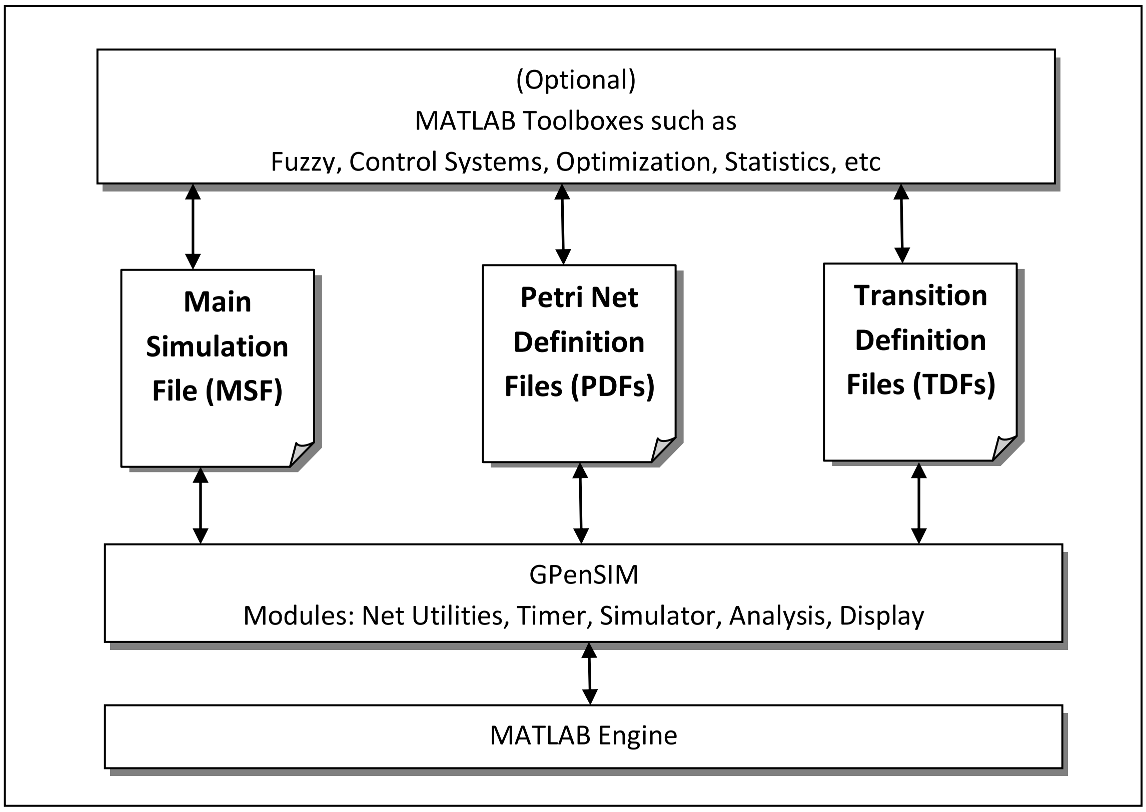

(Optional)
MATLAB Toolboxes such as
Fuzzy, Control Systems, Optimization, Statistics, etc
Main Simulation File (MSF)
Petri Net Definition Files (PDFs)
Transition Definition Files (TDFs)
GPenSIM
Modules: Net Utilities, Timer, Simulator, Analysis, Display
MATLAB Engine

**Figure 2-7: The architecture of GPenSIM of modular components as presented in [2]**

#### 2.5.1.2.1 Petri Net Definition File/s (PDFs)

This file is used to define static Petri Net Graph. Different modules of Petri Net can be defined in various PDFs. This module contains set of places, set of transitions and set of normal & inhabiting arcs with their weights connecting places with transitions. This file is differentiated from other file by adding '_def' at the end of the file name.

#### 2.5.1.2.2 Main Simulation File (MSF)

This is the main file which runs directly from MATLAB platform. It contains initialization for token state & dynamics, firing time etc .This module also includes functions to display and control Petri net properties and results [2]. All the required PDFs are defined in MSF and get loaded in its memory before the beginning of the simulation. MSF is blocked during the simulation process/run, so they don't have any control during simulation. Once the simulation is finished the control comes back to MSF along with results.

#### 2.5.1.2.3 Transition Definition Files (TDFs)

Both pre processors and post processors files are termed TDFs which includes additional conditions defined by users. A pre-processor runs before transition fires and checks whether the enabled transition can fire depending upon the addition conditions where as post-processor runs after the enabled transition fires. This file contains actions to be carried out after firing. TDFs files are called during simulation. MSF don't have control during the simulation however with the use of TDFs, systems can be manipulated during run time. This module works as a guard function in Colored Petri Net. Color values are checked in this file.

There can be a separate pre & post files for each individual transitions and/or a combined common file. For combined preprocessor and post processor the naming convention is 'COMMON_PRE' and 'COMMON_POST' respectively. Whereas for individual transitions, same transition name followed by '_pre' as prefix for pre processor and '_post' for post processor.

### *2.5.1.3 Natural Language interface*

Modeling a discrete event system in Petri Net is a very easy process. User doesn't need to have prior knowledge of Petri Net's mathematics to model. GPenSIM provide users exactly that feature while building a model. It provides a natural language interface. User only needs to identify the basic elements like places, transition, connection between them, initial token etc.

## 2.5.2 Example of GPenSIM in action

Let us take same example used in explanation of Petri net. There to produce a bicycle we need one body frame and two wheels.

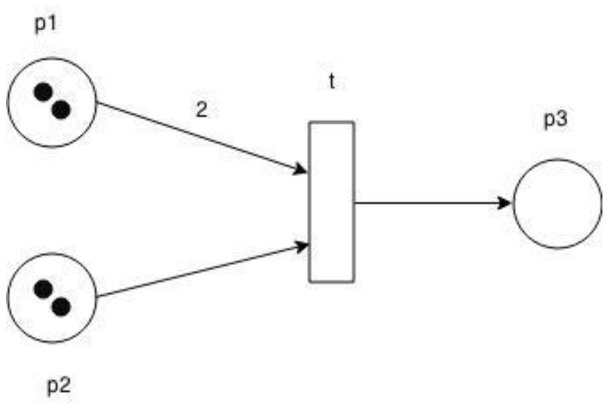


**Figure 2-8: Simple example of bicycle manufacture**

**Step-1: Defining the Petri Net graph**

Petri net graph (model) is defined in PDF. First of all recognize all the places and transitions along with its connecting arcs and their values. These identified elements are entered to the PDF along their related variables.

In the Figure 2-8- There are three places, one transition and three arcs. These information are feed to PDF as shown below.

```
% Example: A simple Classic Petri Net

function [pns] = simple_pn_pdf() ;
pns.PN_name = ‘A bicycle manufacturing’;
pns.set_of_Ps = {‘p1 , ‘p2’, ‘p3’};
pns.set_of_Ts = {‘t1’};
pns.set_of_As = {‘p1’, ’t1’,  2 , ‘p2’, ’t1’,  1 , ‘t1’, ’p3’,  1 };
```

The name to the Petri net model is given by,

> *PN_name = ‘A bicycle manufacturing’;*

Set of places for this Petri net model is passed to PDF,

> *set_of_Ps = {‘p1’, ’p2’, ’p3’};*

Similarly transition name is entered,

> *set_of_Ts = {‘t1’};*

**Step 2: Creating the pre-processor and post processor files**

These files are only necessary if we have to include addition conditions to the transition firing before and after. Since our example doesn’t have any conditions, there is no need for such file/s.

**Step 3: The main simulation file: assigning the initial dynamics**

After defining PDF, now we need is to create a MSF where defined PDF is input as parameter to the function ‘*pnstruct*’.

> *pns =pstruct(‘simple_pn_pdf’);*

In case of more PDFs to include, they can be written as:

> *pns =pstruct('simple_pn_pdf', 'simple_pn_pdf2',...);*

Now, initial dynamics like initial markings (tokens) in different places are assigned to a packet, here the packet is '*dyn*' for this example. Firing time, transition priority and many other dynamic tools can also be initialized by assigning value to the packet. And finally this packet along with defined Petri net graph is forwarded to the function '*initialdynamics*'. '*m0*' a variable of packet represents initial marking.

> *dyn.m0= {'p1', '2', 'p2', '2'};*

> *pni = initialdynamics (pns, dyn);*

Initial dynamics also can have dynamic firing time for the transitions. In used example, let's assume it takes 10 second to assemble a bicycle. So, '*t*' takes 10 sec to fire; it can be coded as follow:

> *dyn.ft = {'t1', 10};*

**Step 4: Simulations and result**

Simulation is processed by the function '*gpensim*'. The values of '*initialdynamics*' are passed as parameter. This functions accesses TDFs during run time.

> *Sim_results = gpensim(pni);*

Here, '*Sim_results*' (output structure) is in the receiving side of the result generated by the simulation. With help of function 'prnss'(meaning print state space) , results stored in the output are displayed in MATLAB command line.

> *prnss(Sim_results);*

Chapter 3

# 3 DESIGN AND METHODOLOGY

The method and design is based on the paper [1]. This paper includes modeling of a human body and later simulating forward kinematics which includes those independent part's with stochastic and deterministic motions [30]. Petri Net and GPenSIM use these modeling of normal human to model and simulate those movements of each part.

According to [1], to model and simulate a normal human body it requires sixteen interrelated limbs. These are further simplified for quicker and easier simulation. It is done by removing the limbs that are smaller in size and doesn't have significant role in the overall movement like limbs in fingers and feet. The line represents a limb with two end points acting as parent and child; for example a forearm is the child of the left arm. Here, the model is developed such that the focus is mostly given to the movement of normal human body rather than the appearance of those limbs. Since each limb's motions are determined by movement of one point attached at the end, therefore all the limbs motions can be simulated concurrently and independently. The main point is to simulate a normal human body as a Discrete Event system (DES). The movement of each limb to get to a new position is represented by the change in value of end point attach of the limb. This is viewed as the discrete event and used for simulation of movement of actual human. When these motions are viewed in 3D space and are governed by forward kinematic, the motion of end points can be transformed in rotational and/or translational ways using the still position as its reference point. The simulated human must be able to move most of its limbs in three different angles with constraint/limit of the angles similar to the normal human beings.

The motions produced by the movement can be categorized into stochastic and deterministic depending upon the pattern. A deterministic motion are the repetitive patterns which are normally deliberate or natural for example a cycle of walking or running along with hand swing which is later descried with the help of gait cycle in this paper. The limbs are also able to move without making repetitive patterns with stochastic motion.

## 3.1 Human body and the motion as discrete event system

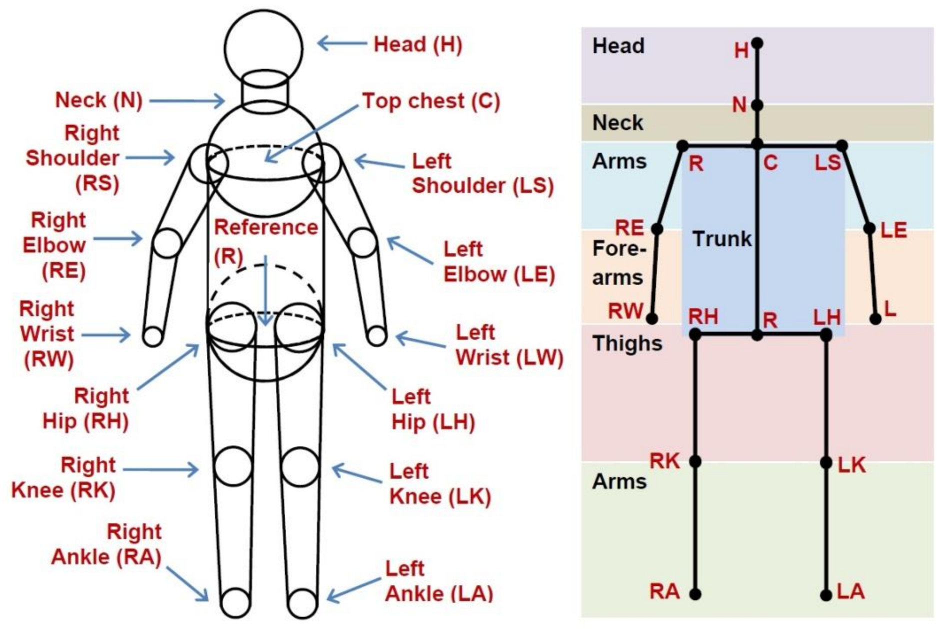
There are lots of papers describing the human models. Depending upon the goals to achieve, the models with matching complexity are considered. The purposed models are widely divided into two areas, computer vision and computer graphics [1]. Computer vision are dedicated to capturing, tracking , analysis and recognizing the human motion through images and videos where as computer graphics mostly relates to simulating/animating human movements as realistic as possible. This paper is solely dedicated to computer graphics. The human motions for simulations are discussed in papers like [31] where focus is given to virtual simulation for better manufacturing processes and, [32], describes the impacts of real world vehicle pedestrian through simulation.

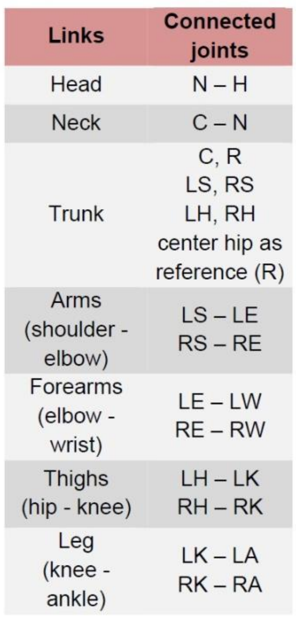
There are seven approaches which can be used to animate human motion as mentioned in [33] and they are “kinematics, dynamics, motion controllers, animation of deformable objects, motion planning, and autonomous character behavior and optimization methods”. The main objective of the paper is to find best way to model and simulate a humanoid robot using Petri Net. To do this first of all the problems and objective that needs to address must be known then figure out the appropriate approach to use.

The position and orientation of the joints determine the state of the human body in kinematics when viewed as a system. The main reason kinematics is widely used to simulate a human body is because the system are taken as discrete event system. Kinematics can be forward or inverse. In forward kinematics, new positions are produced from the transformation matrix which is altered by the change in the angle of the joints. The same procedure is reversed in inverse kinematics where both current and next position is input and transformation matrix is calculated. Calculation performed in inverse are much more time consuming and intense where as in forward it’s more general and simple [34]. In following sections, human body is modeled as discrete event system followed by discussion of human ROM for realistic simulation in 3D space.

## 3.2 A model of human body and range of motion

A simplified version of the frontal side of a humanoid model is shown in Figure 3-1 as described in [35]. This model is a discrete event system consisting of limbs and links to replicate a normal human being. It is made as simple and realistic as possible. Limbs in used are head, neck, trunk, arms, forearms, thighs and legs where head and neck are combined for simplicity. Palms, feet and fingers in hands & legs are not include as they make it less complex and also these silhouette doesn’t have significant roles in the simulation designated to perform according to the objectives. There are separate links which are connected by a joint for each two links except for the ones at the end. The total of sixteen joints is in use for the movement of this model that belongs to the link and moves with the links which are connected with it. The new positions and orientation generated using forward kinematics in 3D space is the result of change in value of joints rather than the limbs. In the figure, sphere/dots represented as joints which are also the centered surface area and cylinder/lines connecting them represented as links.





| Links | Connected joints |
|---|---|
| Head | N – H |
| Neck | C – N |
| Trunk | C, R<br>LS, RS<br>LH, RH<br>center hip as reference (R) |
| Arms (shoulder - elbow) | LS – LE<br>RS – RE |
| Forearms (elbow - wrist) | LE – LW<br>RE – RW |
| Thighs (hip - knee) | LH – LK<br>RH – RK |
| Leg (knee - ankle) | LK – LA<br>RK – RA |

**Figure 3-1: Taken from [1],showing frontal view of a human body's model portrayed as discrete event system using cylinders and spheres connected with each other (Left). The required links and joints for DES of forward kinematics shown using skeleton (middle). The entire list for the skeleton model and their connecting joint (right).**

A robot can be termed as humanoid robot if it has same constrain as that of a human beings. For more realistic and accurate representation of human, a robot must be simulated within the ROM of human motion. The range of motion is given by motion of movable human links which are connected properly to other links through joint covering distance angularly or linearly. The direction of the movement of joint is determined by the relationship of muscles to their axes around which it moves. From [36] , other movements of joints in opposite direction can take place around each axis is given by : bending-extending (flexion- extension) as in elbows , rolling inward-rolling outward (inner rotation – outer rotation) as in shoulders, as well as pushing out- pulling in (abduction-adduction) and forward motion – backward motion (flexion -extension) as in hips. Figure 3-2, shows the normal ROM of normal human links along with movements in opposite direction in 3D space [36, 37]. Three axes are represented as **α**, **β** and **γ** which are further elaborated with minimum and maximum value in Figure 3-3. The values written here are gathered by measuring against the vertical line i.e. 0 using forward kinematics. The values can be positive as well as negative depending upon the possibility of motion.

## 3.3 Forward Kinematics of rigid human body in motion

A rigid body can be modeled as a point since the distance between two points of a rigid body never changes with the movement of the object .So, the motion of rigid body can be represented by motion of a single point. In this paper, a humanoid model is considered as a rigid body and its forward kinematics are illustrated by Figure 3-3[38].

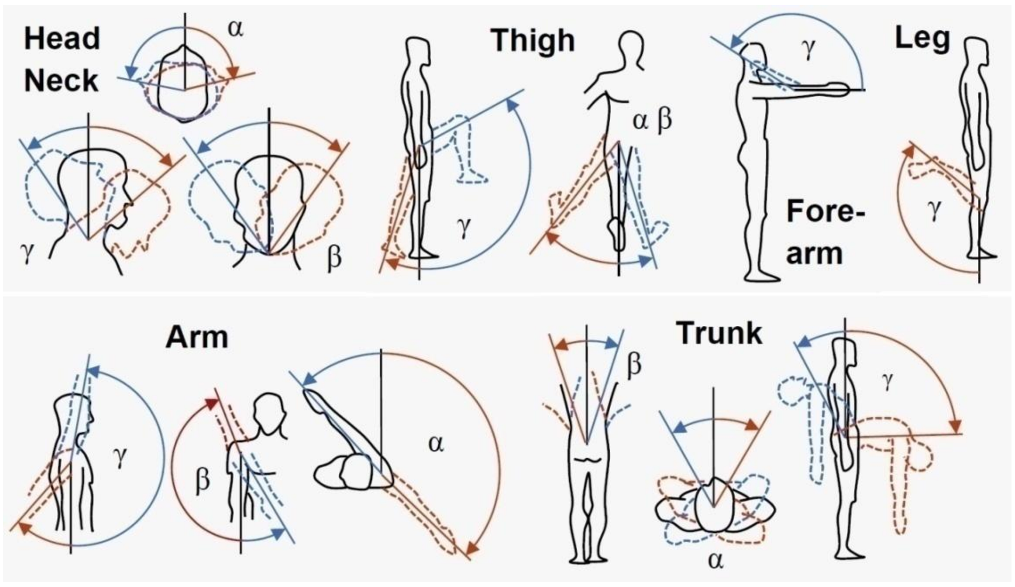
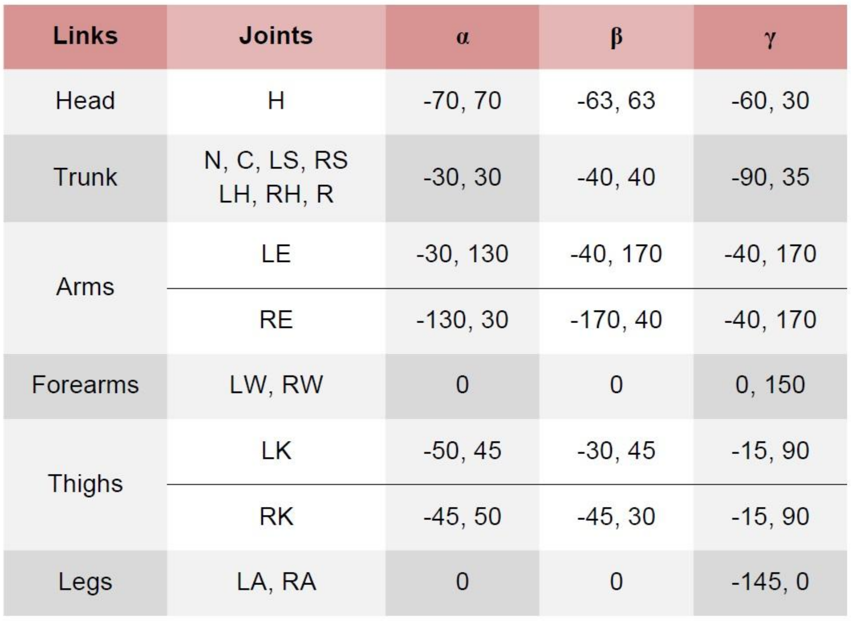


**Figure 3-2: Taken from [1]originally in [36, 37], shows the ROM of human for head, neck, trunk, arms, forearms, thighs and legs. The required angles for forward kinematics for each links indicating different angles are represented by α, β and γ. Angle signs Positive (blue) and negative (red) sum up for opposite movements.**



| Links | Joints | α | β | γ |
|---|---|---|---|---|
| Head | H | -70, 70 | -63, 63 | -60, 30 |
| Trunk | N, C, LS, RS LH, RH, R | -30, 30 | -40, 40 | -90, 35 |
| Arms | LE | -30, 130 | -40, 170 | -40, 170 |
| | RE | -130, 30 | -170, 40 | -40, 170 |
| Forearms | LW, RW | 0 | 0 | 0, 150 |
| Thighs | LK | -50, 45 | -30, 45 | -15, 90 |
| | RK | -45, 50 | -45, 30 | -15, 90 |
| Legs | LA, RA | 0 | 0 | -145, 0 |

**Figure 3-3: Taken from [1] originally in [36, 37], Demonstrates forward kinematics of normal human body as rigid model (top). The range of movement (minimum and maximum) for the purpose of simulation of angles α, β and γ in degree are listed (bottom).**

## 3.4 Deterministic Human motions with Human gait cycles

In the paper [30], which this paper is based on, discusses two types of gait denoting the pattern of walking and running which are represented by parameter g=1 and g=2 respectively. The periodic repetitions of those patterns of normal human walking and running are termed as gait cycles. Human body can move independently irrespective of their physical connection. The body can be divided into upper and lower parts. Upper part includes head, neck, trunk, arms and forearm where as rest are considered as lower arms. Since lower half has direct contact with the earth, the repetitive motion is easy to analyze while walking and running so gait cycle is only applied to lower half unlike upper half which do make movement while walking and running but doesn't follow a fixed pattern.

A gait cycle starts with a foot on the ground and ends with same foot getting back on the ground. This event is termed as initial contact (IC). Stance and swing are two phases that make up both running and walking gait cycle. Stance begins with IC and finishes with toe getting off the ground which is called as toe off (TO). Swing phase begins with event TO and ends at IC. Figure 3-4 describes the gait cycles in detail [39] .

In figure, a gait cycle of both left and right feet is shown in a detailed fashion where 60% stance and 40% swing are shown during running and approximately 35% stance and 65% swing while running [39, 40]. The main difference between walking and running , other than those difference in percentage between stance and swing is, during running both feet overlap at some point of stance where as during running at some point both feet are on the air. The

Figure 3-2, displays three important angles: the thigh extension x, the thing flexion w and the leg flexion which corresponds to hop extension, hip flexion and knee flexion respectively. The normal values of angles obtained from [39, 40] with deep measurements are used in famous mathematical functions for simulation.

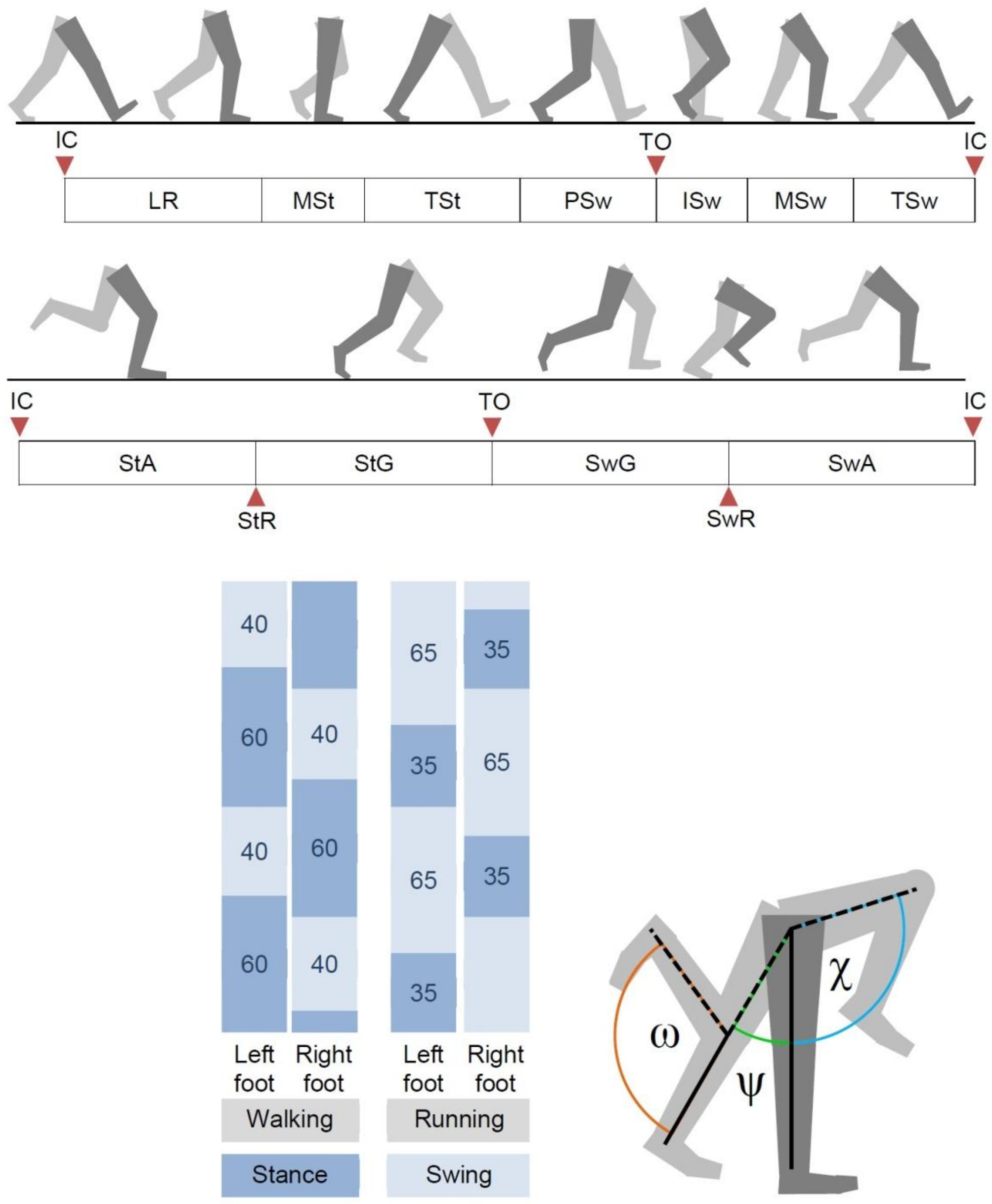


**Figure 3-4: Taken from [1]originally in [39], (a) the full cycle of gait for walking (top) and running (below) of a normal human with components as sub phases: initial contact (IC), toe off (TO), loading response (LR) , midstance (MSt), terminal stance (TSt), preswing (PSw), initial swing (ISw), midswing (MSw), stance reversal (StR) and swing reversal (SwR). (b) Comparing gait cycle of running and walking for stance and swing duration of both feet (left). The thighs extension x, the thigh flexion w and the leg flexion w are three important angles used in simulation to generate gait cycle (right).**

| Joint set 1 | Joint set 2 | γ walk, forward | γ walk, backward | γ run, forward | γ run, backward |
|---|---|---|---|---|---|
| LE | RE | -10 | 10 | -30 | 5 |
| LW | RW | 10 | 20 | 90 | 120 |
| RK | LK | 20 | -20 | 25 | -25 |
| RA | LA | -5 | -15 | -25 | -35 |

**Table 3-1: The ROM of degree obtained from simulating gait cycle along with hand swing using LI [1]**

While walking and running, the motions starts from the lower part of the body where most the contributing are done by the thighs and legs .So, to decrease the complexity ,low significant feet and finger are not included. But to balance the body the upper part also starts to come in motion. The motion of upper part may vary from person to person. It gets very difficult to generalize mathematically, that is; the swinging hands during walking and running. This is the reason for the requirement for close inspection of gait cycles with hand swing for Discrete Event System. Gait cycle along with hand swing involves two set of limbs: [LS, LW, RK, RA] and [RS, RW, LK, LA] .When monitored closely, it is observed that when one set of limbs move forward, other set moves backward. This is an example of stochastic motion where ROM for both set requires minimum and maximum angles.

Table 3-1, shows the value of all the sets of ROM for knees and ankles along with arms and wrist for hand swing which corresponds to the Figure 3-5. Basically Figure 3-5, is the resulting plots of the ROM for walking and running which is generated by three angles in a normal human gait cycles take from [39, 40] and Fourier parameter values.

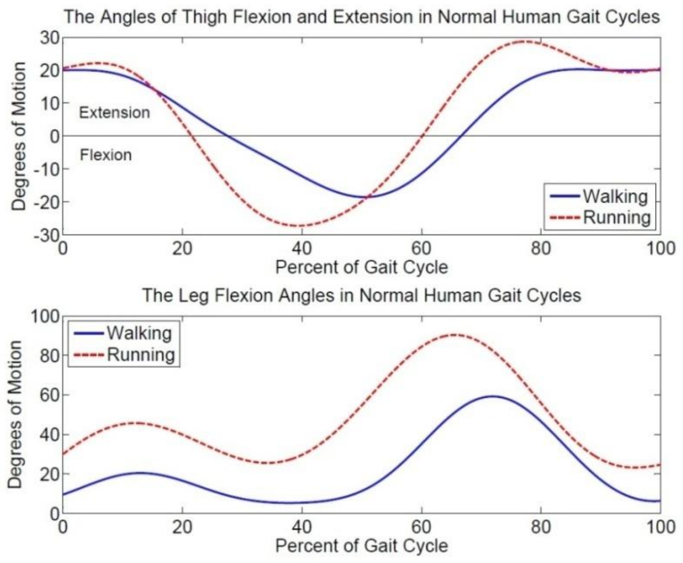


**Figure 3-5: Taken from [1], ROM of thighs representing gait cycles of a healthy human for walking and running .**

## 3.5 Designing Humanoid robot in GPenSIM

The above design of a normal human presented as Discrete Event System is used for creating and modeling robot for simulation in GPenSIM. GPenSIM is a general purpose Petri Net simulation tool. In Petri net, there are transitions and places. They are interconnected with arcs. Token are exchanged through transitions along arcs between places. These flowing of tokens represent simulation for discrete events. Concurrent and dynamic[2] systems can be modeled and simulated easily with this concept. Due to such features Petri Net is considered. The main objective of this paper is to model and simulate humanoid robot using this concept.

The human model discussed here consists of sixteen links and limbs shown in Figure 3-1 above. To decrease the complexity, it only includes head, neck, trunk, arms, forearms things and legs but excludes fingers, feet and palms. The links are connected using joints. This model is made of sixteen essential joints joining links for movement. Forward kinematics is used for simulating the position and orientation of the model; hence preference is given to the joint rather than links. To achieve more realistic and accurate simulation of human, the movement of robot must be constrained similar to that of human movement or also viewed as ROM of human. ROM (Range of Motion) can be defined as limited angular or linear displacement that can be achieved by human link with in its motion. These links are properly connected to other links via joints. The ROM of human of different part of human body is shown by the Figure 3-2. The maximum and minimum values (degrees) of the ROM of human for different axes **α**, **β** and **γ** with respect to forward kinematics are given in Figure 3-3. In the table of Figure 3-3, 0 degree indicates as not possible to move in that particular direction and negative sign refers to the counter direction.

The movements of all sixteen joints are independent of each other. Furthermore, different possible axes of any joint can also move not depending on other axis. For example, a ball can be kicked sideways or straight. Utilization of only one axis of joint is enough for kicking straight but in case of kicking sideways, more than one axis needs to come in action. In this implementation, different axes can be used together or separately. They don't depend on each other. So, a separate Petri net model is implemented for each joint and each axes for concurrent movement. A model for representing an axis of a joint is shown in Figure 3-6: The Petri net model for a joint in any axis. Here the model has three places along with four transitions. Normal arcs with unity weights are positioned to connect places with transitions and maintain a flow. Also included are inhabiting arcs. Each degree value of joint is represented by a token.

There are three places; '*p_al*', '*p_pos*' and '*p_neg*'. At starting phase, '*p_al*' holds all the tokens. The total number of token denotes total degree the particular joint can move. At beginning, joints are in still position, so all tokens reside in '*p_al*'. Places '*p_pos*' or '*p_neg*' receives tokens depending upon the movement, if it's in positive direction or counter direction respectively. At any given instance, the number of tokens in '*p_pos*' or '*p_neg*' represents the angle value of that particular joint. Note that, both places can't have tokens at the same time since only one angle value is possible. This is achieved with the help of inhibiting arcs.

Transitions represent events and movements of joints are events. Around 4 transitions are used namely ‘*t_pos*’,’*t_neg*’, ‘*t_pos_back*’ and ‘*t_neg_back*’. For movement in positive direction, transition ‘*t_pos*’ fires and takes tokens from ‘*p_al*’ and places it in ‘*p_pos*’ . Similarly for negative direction, ‘*t_neg*’ fires which results in depositing tokens to ‘*p_neg*’ from ‘*p_al*’. But ‘*t_pos*’ can’t fire if there is any token in ‘*p_neg*’ and same with ‘*t_neg*’; if there is presence of token in ‘*p_pos*’. Inhibiting arcs are used in both transitions to prevent from firing in such situations as shown in Figure 3-6: The Petri net model for a joint in any axis. Also some more inhibiting arcs are being implemented in both transitions to limit in case of over flow of token which could have resulted in crossing the maximum angle value. When the joint is in positive or negative degree and has to make a move to opposite direction, transition ‘*t_pos_back*’ or ‘*t_neg_back*’ is utilized. These transitions take tokens back to place ‘*p_al*’. To shift from positive to negative degree value or vice versa. Firstly, all tokens must return back to place ‘*p_al*’ and finally transition ‘*t_pos*’ or ’*t_neg*’ is fired depending upon the shift from positive to negative or negative to positive.

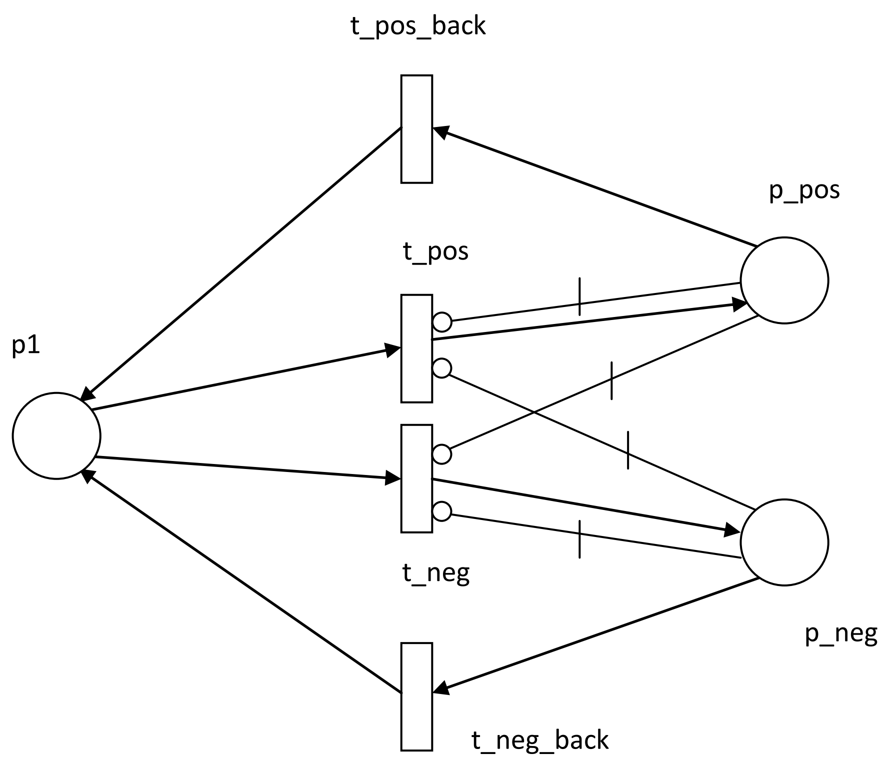


**Figure 3-6: The Petri net model for a joint in any axis**

| Element Name | Element | Designated Work |
|---|---|---|
| p1 | Place | Holds tokens as resting place |
| p_pos | Place | Holds tokens denoting positive degree joint position |
| p_neg | Place | Holds tokens denoting negative degree joint position |
| t_pos | Transition | Fires tokens from ‘p1’ to ‘p_pos’’ indicates movement of joint to positive direction |
| t_neg | Transition | Fires tokens from ‘p1’ to ‘p_neg’; indicates movement of joint to negative direction |
| t_pos_back | Transition | Fires tokens from ‘p_pos’ back to ‘p1’; indicates recoiling movements back from positive degree |
| t_neg_back | Transition | Fires tokens from ‘p_neg’ back to ‘p1’; indicates recoiling movements back from negative degree |

**Table 3-2: Work details of individual elements of model**

This Petri net model is used for a single axis for each joint and has maximum and minimum values. They can be grouped together to present in a broader way like right hand module can have joints of right elbow and right wrist. Nearly all joints have three separate axes. A single module of one joint includes Petri Net model for each axis. The combination of all modules of joints will create a fully functioning humanoid robot. In this way a humanoid robot can be modeled and simulated using Petri Net and GPenSIM. For multiple robots, separate set of joints can be created under separate module for different robots. The main aim for this simple design was to generate uncomplicated simulation and later expand it to swarm robots. The design is discussed below.

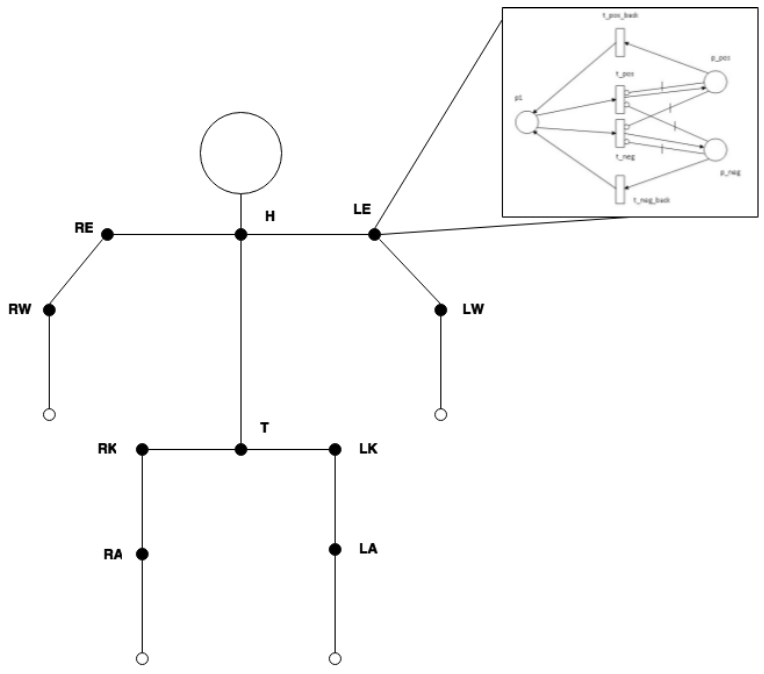


**Figure 3-7: Petri Net model used in  joints**

In Figure 3-7, a robot is shown with joints denoted as small circles and lines as limbs. Only black circles are used in the robot for this project. These joints are designed to move limb that is connected to it. All the joints are actually composed of the separate Petri Net model as shown in the Figure 3-7 for joint ‘*LW*’. This figure only illustrates movements of joints in only one single axis.

The proposed model of humanoid robot is defined in 3D space. So movements to all three axes are possible. Same Petri Net models are used separately for all possible axes. Except for joints ‘*RW*’,’*LW*’, ‘*RA*’ and ‘*LA*’, all other joints have movement in all three axes. The **Figure 3-8** below displays separate models designed to make movement in three different axes for joint ‘*LE*’. They are totally independent from each other and controlled by an instructor which can be interrelated as mind of robot.

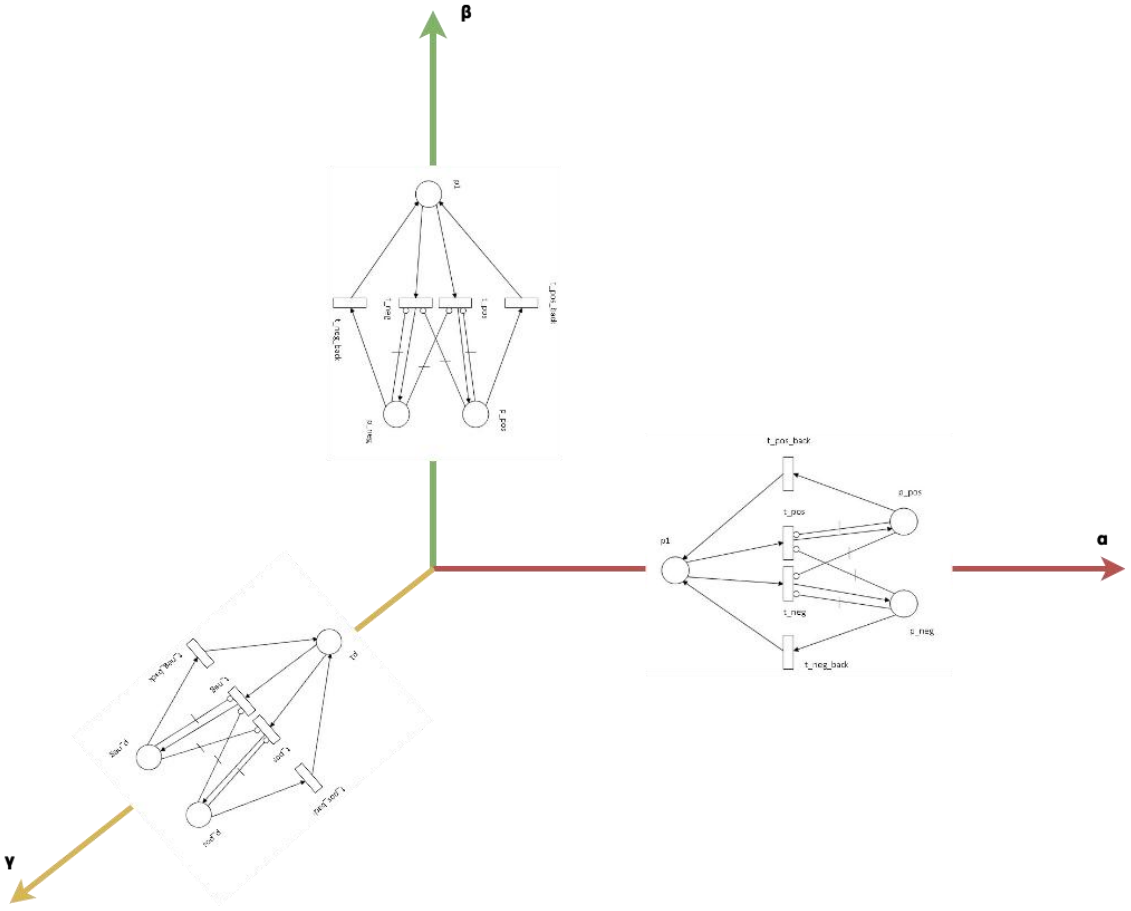


**Figure 3-8: Illustration of all possible axes in a joint**

These models are controlled through a transition termed as instructor. This works as mind of robot which manipulates all the movement of joints. The movements in this work are controlled by the user with help of a file. This is explained in detail below in next section. The **Figure 3-9** below illustrates the connection between the instructor and all joints.

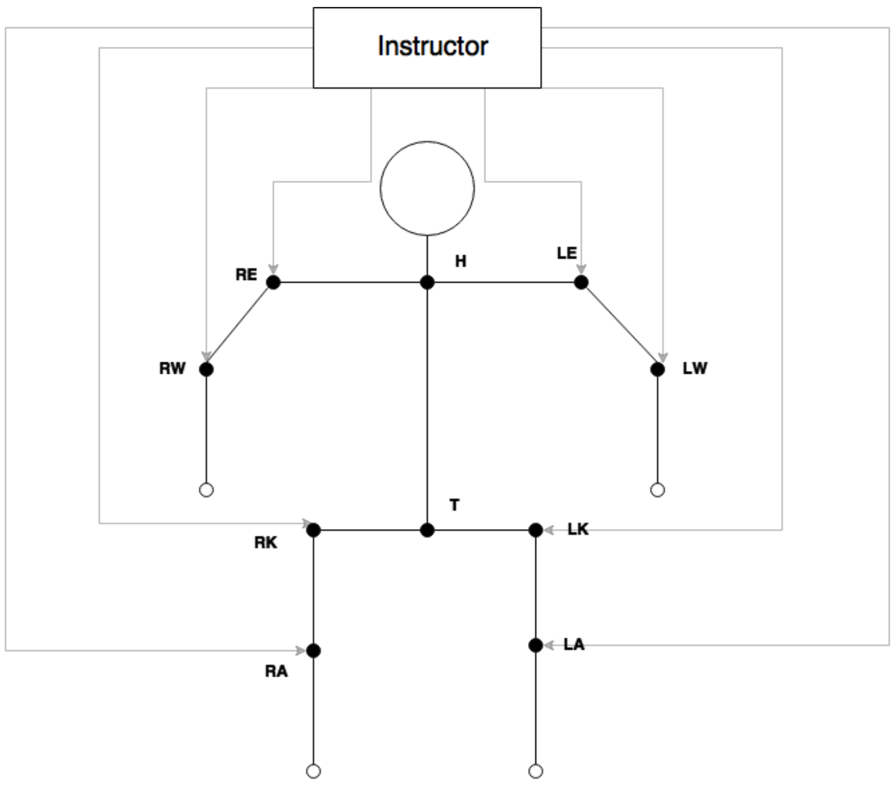


**Figure 3-9: Instructor controlling all joints**

This design is later carried out for multiple robots for swarm setting. For such environment, in this project a main instructor is constructed. This head instructor is connected to all personal instructors of each individual robot. This design shows master slave architecture. Every instructor of robots only orders movement of joints in accordance to instruction received from the head commander. This keeps all robots synchronized and maintains swarm property. This is illustrated in a **Figure 3-10** below.

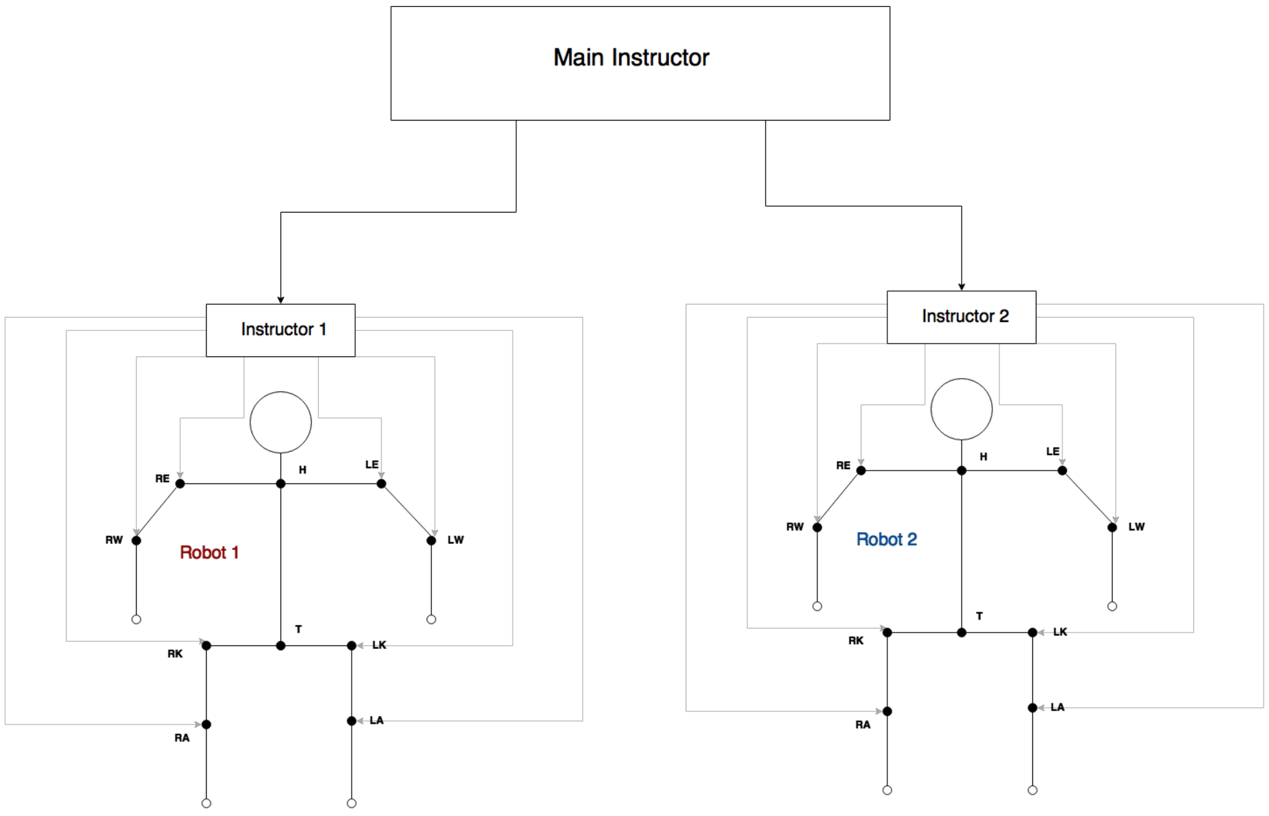


**Figure 3-10: Multi robots in swarm setting**

In this way, a model of a humanoid robot was designed in Petri Net for the implementation in GPenSIM for simulation. Also this model was successfully expanded and designed for use in swarm setting where multiple robots co-ordinate in a synchronous manner with a main instructor acting as a coordinator to individual instructors working as mind of robot. The main instructor reads movement instructions from a file input by user. The implementation details are explained in next section of this paper.

Chapter 4

# 4 IMPLEMENTATION

The design of a humanoid robot with sixteen joints as described in the design section is implemented in Petri Net using GPenSIM. The design model for Petri net representing human joint is includes as well. The considered human model follows forward kinematics so the resulting positions and orientations belong to the joints rather than links. The movements of joints have restrictions which defines a robot as humanoid robot. For the simulation of this robot, there are sixteen joints that can move independently from each other. Along with all mentioned joints, their axes are also considered. Most of these joints have 3 axes denoted by **α**, **β** and **γ**. These axes also have their own movement and don't have big impact on others. So, are also seen as an independent entity.

## 4.1 Design Model in GPenSIM

GPenSIM a MATAB tool is used here to model Petri Net designs for joints and put them together to form a fully functioning humanoid robot. All the codes are written and run in MATLAB. The main simulation file (MSF) for the robot is named as '*robot.m'*. This consists of all the initialization for all values related to the joints and their axes. There is separate Petri Net Definition files (PDFs) defined in MSF. Each joint is treated as a single module, so every joint have separate PDF. All the different axes are bounded within the section of each joint. The joints N, C, LS, RS, LH, RH and R are viewed as a single joint T which is Trunk. The main focus of the simulation is given to the movement such as walking and running where those joint don't play a big part, so they are considered as a single joint. This helps to reduce calculation time for simulation till some extend. The PDF for each joint is named as '*head_pn_pdf*' for Head (H) , '*trunk_pn_pdf*' for Trunk (T) , '*LeftElbow_pn_pdf*' for Left Elbow (LE) , '*RightElbow_pn_pdf*' for Right Elbow (RE) , '*leftWrist_pn_pdf*' for Left Wrist (LW), '*rightWrist_pn_pdf*' for Right Wrist (RW) , '*LeftKnee_pn_pdf*', '*RightKnee_pn_pdf*' for Right Knee (RK) , '*leftLegs_pn_pdf*' for Left Ankle (LA) and '*rightLegs_pn_pdf*' for Right Ankle (RA). A Petri Net model for a head is shown below followed by its implementation in GPenSIM using PDFs, MSF and TDFs.

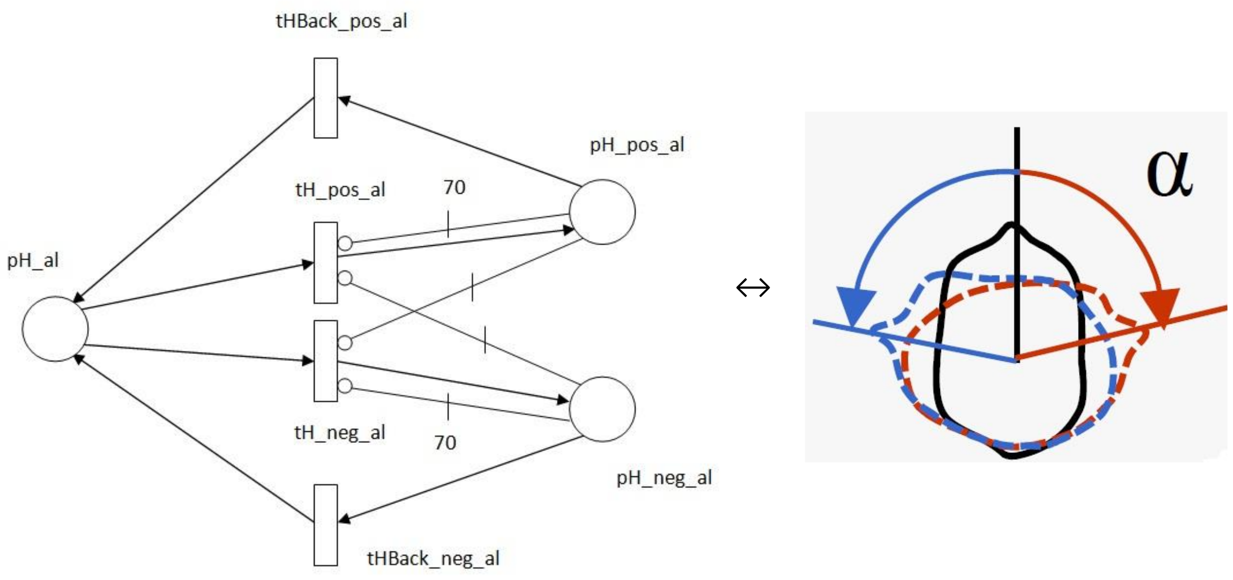


**Figure 4-1: This is Petri Net model used for α angle in Head joint. The maximum degree for the movement in both is 70 in both directions.**

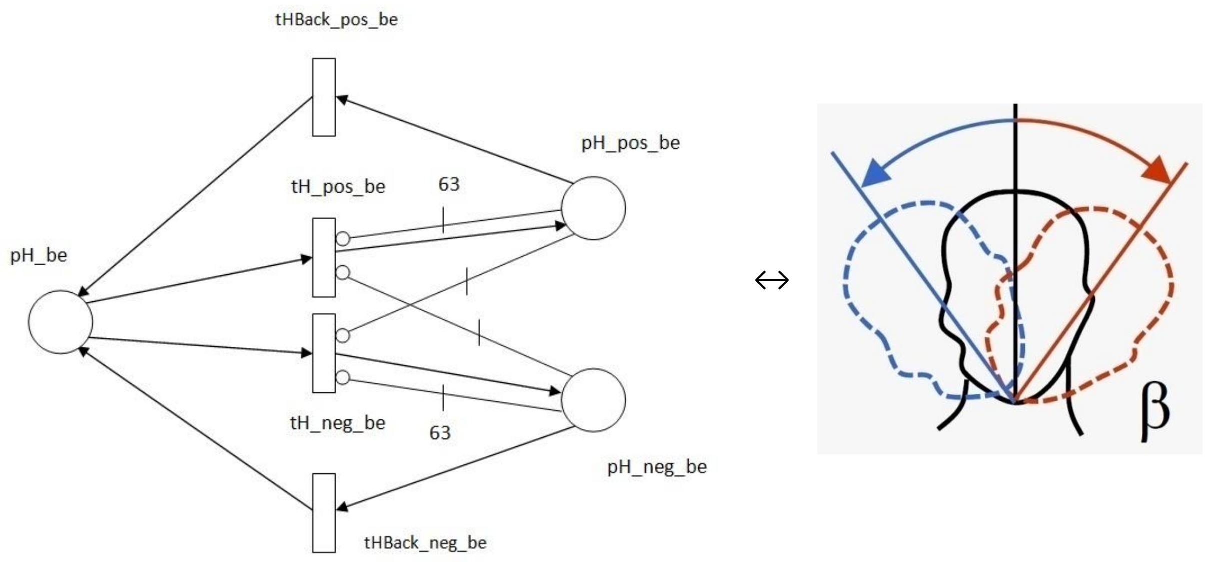
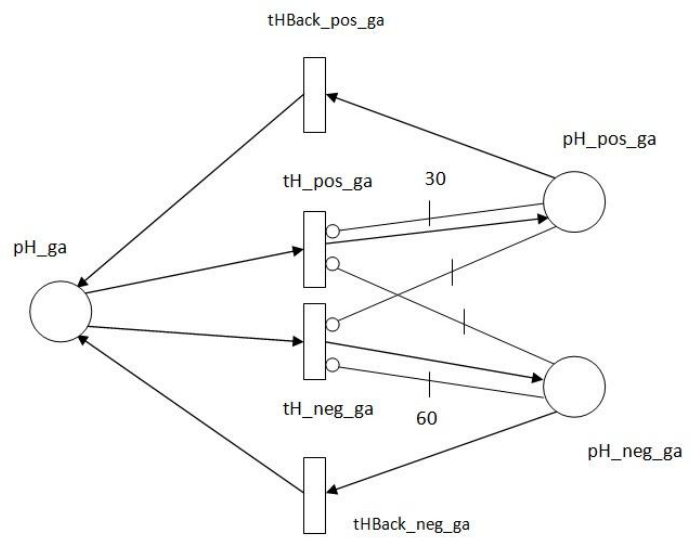


**Figure 4-2: This is Petri Net model used for β angle in Head joint. The maximum degree for the movement in both is 63 in both directions.**



↔

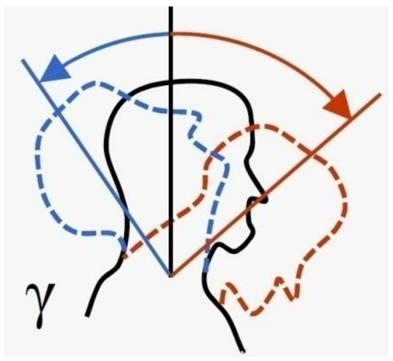


**Figure 4-3: This is Petri Net model used for γ angle in Head joint. The maximum degree for the movement in positive direction is 30 and opposite is 60.**

## 4.2 Petri Net Definition File (PDF) for model

For the models shown above, each model has three places and four transitions connected by arcs weighing one and four inhibiting arcs where two of these inhibiting arcs weight one and other two weigh equal to maximum possible value of the joint. The models shown above are different angles of joint Head which is defined in one PDF as shown below under name: '*head_pn_pdf*'.

```
%Petri Net GpenSim
function [pns] = head_pn_pdf()

    pns.PN_name= 'The head of the robot';
    pns.set_of_Ps = {'pH_al','pH_pos_al','pH_neg_al',...
                     'pH_be','pH_pos_be','pH_neg_be',...
                     'pH_ga','pH_pos_ga','pH_neg_ga'};

    pns.set_of_Ts = {'tH_pos_al','tHBack_pos_al','tH_neg_al','tHBack_neg_al',...
                     'tH_pos_be','tHBack_pos_be','tH_neg_be','tHBack_neg_be',...
                     'tH_pos_ga','tHBack_pos_ga','tH_neg_ga','tHBack_neg_ga'};

    pns.set_of_As = {'pH_al', 'tH_pos_al',1 ,'pH_al', 'tH_neg_al',1 ,...
                    'tH_pos_al', 'pH_pos_al',1 ,'tH_neg_al', 'pH_neg_al',1 ,...
                    'pH_pos_al', 'tHBack_pos_al',1 ,'tHBack_pos_al', 'pH_al',1 ,...
                    'pH_neg_al', 'tHBack_neg_al',1 ,'tHBack_neg_al', 'pH_al',1,...
                    ...
                    'pH_be', 'tH_pos_be',1 ,'pH_be', 'tH_neg_be',1 ,...
                    'tH_pos_be', 'pH_pos_be',1 ,'tH_neg_be', 'pH_neg_be',1 ,...
                    'pH_pos_be', 'tHBack_pos_be',1 ,'tHBack_pos_be', 'pH_be',1 ,...
                    'pH_neg_be', 'tHBack_neg_be',1 ,'tHBack_neg_be', 'pH_be',1 ,...
                    ...
                    'pH_ga', 'tH_pos_ga',1 ,'pH_ga', 'tH_neg_ga',1 ,...
                    'tH_pos_ga', 'pH_pos_ga',1 ,'tH_neg_ga', 'pH_neg_ga',1 ,...
                    'pH_pos_ga', 'tHBack_pos_ga',1 ,'tHBack_pos_ga', 'pH_ga',1 ,...
                    'pH_neg_ga', 'tHBack_neg_ga',1 ,'tHBack_neg_ga', 'pH_ga',1 };

    pns.set_of_Is ={'pH_pos_al', 'tH_pos_al',70 ,'pH_pos_al', 'tH_neg_al',1 ,...
                    'pH_neg_al', 'tH_neg_al',70 ,'pH_neg_al', 'tH_pos_al',1,...
                    ...
                    'pH_pos_be', 'tH_pos_be',63 ,'pH_pos_be', 'tH_neg_be',1 ,...
                    'pH_neg_be', 'tH_neg_be',63 ,'pH_neg_be', 'tH_pos_be',1,...
                    ...
                    'pH_pos_ga', 'tH_pos_ga',30 ,'pH_pos_ga', 'tH_neg_ga',1 ,...
                    'pH_neg_ga', 'tH_neg_ga',60 ,'pH_neg_ga', 'tH_pos_ga',1};
```

The Figure 4-1, Figure 4-2 and Figure 4-3 shows the Petri Net model being used for head in angles **α**, **β** and **γ** respectively. Initially the maximum number of possible value either in positive or negative direction. This is identified and used as total number of token in place '*pH_al*', '*pH_be*' and '*pH_ga*'. The tokens represent the degrees. Since the maximum possible degrees of movement are known, it becomes convenient to use as many token as reachable degree values. As described in the design section, here for **α** angle; '*pH_pos_al*' replicates as positive transition which holds tokens that reflect positive degree movement and number of token represent value of the degree in positive direction. This is same in case of '*pH_neg_al*' as negative place holder where token in that place represents negative degree. Similarly, the working mechanism for both **β** and **γ** angle is same.

All the arcs linking places with transitions have unit weight except two inhibiting arcs. There are four inhibiting arcs in each model. In Figure 4-1, inhibiting arc stretching out from '*pH_pos_ga*' to '*tH_pos_ga*' has weight of 60 indicating transition '*tH_pos_ga*' is only enabled if total number of token in place '*PH_pos_ga*' is less 60. This is a blocking mechanism for controlling tokens (degree value) from exceeding more than maximum capable value. Here maximum reachable value for both positive and negative is 60 so both arcs weigh same i.e. 60. It is same for angle **β**. But for angle **γ**, inhibiting arcs weighs are different as maximum positive value is 30 and opposite value is 60. The other inhibiting arcs weighting one are also for controlling. These arcs blocks positive transitions from firing if there is any token in negative place and vice versa. This is implemented in all the models. Blocking mechanism can be used in coding part too but here it is hard coded in the model because this always remains same and doesn't have to change time and again. These restrictions make a robot, a realistic humanoid robot.

## 4.3 Transition Definition File (TDF) for model

The conditions for and after firing of transitions are written in these Transition definition files (TDFs). There can be common TDF for all transitions under the name 'COMMON_PRE' for conditions to enable transitions and 'COMMON_POST' for actions that need to be carried out after transitions are fired. A separate file can also be used for each individual transition with same file name followed '_pre' as suffix for pre conditions and '_post' for post conditions. In this implementation we are using separate files as shown in Figure 4-1, for pre conditions because there are lots of transitions and putting everything together will make it hard to manage. Each model has four transitions as described in design part. The transitions used here for head's **α** angle are *tH_pos_al*' , '*tH_neg_al*' ,'*tHBack_pos_al*' and '*tHBack_neg_al*'.

Further discussing with Figure 4-1, Transition '*tH_pos_al*' can be viewed as positive transition which is enabled when total number of token denoting total value of degree in place '*pH_pos_al*' is less than entered degree value for **α** angle of head joint. This condition is written in pre condition file for this transition as follows:

```
global global_info;
b1 = get_place('pH_pos_al');

if and(gt(global_info.H_angle(1),0), lt(b1.tokens,global_info.H_angle (1))),
        fire = 1;
else
        fire = 0;
end
```

Here, '*global_info*' is a packet which is used to access and exchange global parameter values. '*global_info.H_angle*' is a global cell array of 1 × 3 dimensions which is used to represent input degree of HEAD (H) joint. The first value of the cell corresponds to angle **α**, second value to **β** and the third value to **γ**. Since Figure 4-1 deals with angle **α**, first value of cell variable is used as '*global_info.H_angle(1)*'. '*get_place*' function returns total number of token located in place entered as parameter. In this case place '*pH_pos_al*' is set as parameter for the function. So, this transition fires only when the input value of degree is greater than 0 and total number of tokens in the linked place is less than the input value. 'Greater than 0' condition is to check if input value is positive. The transition fires until the number of tokens equal to the degree entered. At each firing, only one token is fired at a time since the arc weight is 1 between the transition and places. This represents that the robot can move only one degree at one simulation time. However for negative firing; small change has been made, to check if it is value input is negative else than that it is similar to positive firing.

The remaining two transitions '*tHBack_pos_al*' and '*tHBack_neg_al*' are enabled to send tokens back to place '*pH_al*' from places '*pH_pos_al*' and '*pH_neg_al*' respectively. This relates to change in degree value. If the new input value is less than previous positive value or greater than previous negative value these transitions are enabled correspondingly to maintain the token same as the input value. These conditions are checked in those pre processors for transition. Conditions for '*tHBack_pos_al*' is as follows:

```
    global global_info;
    b1 = get_place('pH_pos_al');

    if and(gt(b1.tokens,global_info.H_angle(1)),ge(global_info.H_angle(1),0)),
            fire = 1 ;
    elseif le(global_info.H_angle(1),0),
            fire = 1;
    else
            fire=0;
    end
```

Here the definition of all the variables remains same. The condition to check is reversed from that of positive transition. This transition is enabled when the input value is greater than 0 and number of tokens in place ‘*pH_pos_al*’ is greater than input value meaning the robot has to decrease some degree from that joint as it is currently in higher degree. It is also enabled if the movement has to be made in negative degree. For this all tokens have to be sent back since for negative angle movement, there should not be any tokens in positive place. Similar case in ‘*tHBack_neg_al*’ where this transition is enabled; if there are tokens present in negative place and positive degree movement has to be made and also for lower negative value. These transitions send token back to main place ‘*pH_al*’.

The variable timings for the firing of transitions are also implemented. Normally when performing a task more than one joints come in action. These joints don’t have direct affect on other joints in terms of their movements but to get things done they need to have co ordination amongst them.  This leads to variable timing of joint movement. To achieve same degree of motion in different time, some joints might take longer time than other. This can be set up as input by the user alongside degree movement value. Following lines of code are used for this.

```
ctime = current_time();

modul = mod(global_info.r1_var_LKp(1),global_info.r1_LK_angle(4));
      if and(le(modul,0),eq(fire,1)), %if fire is true and module < 0
            fire = 1;
            global_info.r1_var_LKp(1) = 0;
      else
          fire = 0 ;
      end
```

Here, function ‘*current_time*’ gives current simulation time when invoked. Variable ‘*global_info.var_LKp(1)*’ is first cell of the array which is used as a counter. This counter is increased as per number of times enabled, fire when reaches firing time assigned by user and sets back to 1 when fired. ‘*gobal_info.LK_angle(4)*’ holds user input for variable timing. For example, for one degree movement in two simulation time, user should input value two to the firing time variable. This variable is compared with counter to maintain the firing time and also checks if enabled. When those conditions are true, it fires and resets counter.

## 4.4 Main Definition File for model

As mentioned above, this is named as '*robot.m*'. This contains all the initialization of the variables used as joints and their axes along with counter for variable firing time. A joint in this implementation is represented as a cell array. This array consist of four cells where the first is used for **α** axis, second for **β**, third for **γ** and finally last is for firing time . They are named using their name initials followed by '_angle'.

For Head joint, global_info.H_angle = [0 0 0 0];

All values are initialized to zero to portray robot is in initial stage with no movements. This can also be done later during simulation using function 'initialize ()'.

For better simulation results, sampling time used here is 0.5, which is inserted as

```
global_info.DELTA_TIME= 0.5; %timer increment value is 0.5 sec(high
sampling rate)
```

As we discussed before, instead of using all sixteen joints, only ten joints are implementation. The main focus of this robot is to perform stochastic and deterministic movements. For deterministic we are only considering walking and running using gait cycle which doesn't involve all the joints. To decrease the complexity of robot, joints neck (N), Chest(C), Left Shoulder (LS), Right Shoulder (RS), Left Hip (LH), Right Hip (RH) and Reference(R) are viewed as a single joint Trunk (T). So total ten joints are defined in PDF separately and their name are passed in MSF along with a PDF to '*pnstruct*'. This is shown below:

```
pns = pnstruct({...
                'instructor_pn_pdf',...
                 'head_pn_pdf','trunk_pn_pdf',...
                'LeftElbow_pn_pdf','RightElbow_pn_pdf',...
               'leftWrist_pn_pdf','rightWrist_pn_pdf',...
                'LeftKnee_pn_pdf','RightKnee_pn_pdf',...
                'leftLegs_pn_pdf','rightLegs_pn_pdf'});
```

Total number of token given to an axis of joint meaning maximum degree it can cover during movement. This stat is obtained from forward kinematics illustration shown in table of Figure 4-1. This gives maximum and minimum ranges for all three axes **α**, **β** and **γ**. These ROM for simulating joints are input as follows where '*pH_al*' represents place which holds tokens for **α** axis of head joint.

```
dyn.m0 = {...
            'pH_al' , 70, 'pH_be' , 63,'pH_ga' , 63,...
            'pT_al' , 30, 'pT_be' , 40,'pT_ga' , 90,...
            'pLE_al' , 130, 'pLE_be' , 170,'pLE_ga' , 170,...
            'pRE_al' , 130, 'pRE_be' , 170,'pRE_ga' , 170,...
            'pLW_ga' , 150,...
            'pRW_ga' , 150,...
           'pLK_al' , 50, 'pLK_be' , 45,'pLK_ga' , 90,...
           'pRK_al' , 50, 'pRK_be' , 45,'pRK_ga' , 90,....
           'pLA_ga' , 145,...
           'pRA_ga' , 145};
```

Finally after the simulation, the result is displayed using two figures. Each figure contains set of joints. As simulation is done for walking and running, the focus is given to **α** and **γ** angle of the joints. The figures that are being displayed are only for **α** and **γ** angle of the joint used for movement. These set of joints are grouped according to the table of the ROM of degree obtained from simulating gait cycle along with hand swing. First set consist Left Knee (LK), Left Ankle (LA), Right Elbow (RE) and Right Wrist (RW) where as second set consist Right Knee (RK), Right Ankle (RA), Left Elbow (LE) and Left Wrist (LW). This is done as follows:

```
 figure(1),plotp(sim,{ 'pLK_pos_al', 'pLK_neg_al','pLA_neg_ga',
'pRE_pos_al', 'pRE_neg_al', 'pRW_pos_ga'});
         figure(2),plotp(sim,{ 'pRK_pos_al', 'pRK_neg_al','pRA_neg_ga',
'pLE_pos_al', 'pLE_neg_al', 'pLW_pos_ga'});
```

## 4.5 Instruction processing

The instructor used here is an independent transition. This fires at every simulation time as it is always enabled. So commands are given to robot as pre condition through this transition using global packet that is '*global_info*'. This TDF checks user input file in serial way and executes according to the time value in the time column of the file's selected row. The structure of the input file is described later in this paper. Using the gait chart and value defined in Table 3-1 of design section, the motion of walking and running is implemented in robot through repetitive looping command given through instructor file. User can also input stochastic command like move elbow ten degrees at time 20 or so on in the file. These stochastic and deterministic movements are differentiated by names as task 1 /task 2 etc. For current implementation task 1 represents walking, task 2 represents running and task 3 represents stochastic movement.

The movement is achieved by changing the variable value which represents individual joints. For example, to move **α** angle of head joint to 5 degree, assume it is in initial stage where its value is 0 degree. Its only enough to change the cell value representing that axis to 5. The firing time given by the end cell of variable cell array can also be altered according to need. 1

or less than 1 meaning normal firing where as in case of more than one, firing takes place as one degree/token move/fire per input firing time value.

```
global_info.H_angle = [5 0 0 1];
```

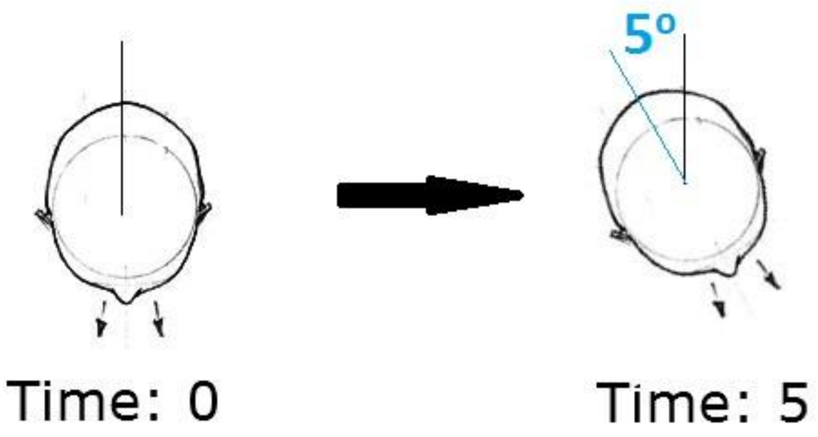


**Figure 4-4: Demonstration of Head movement to $5^o$ in postive direction given by example above.**

In this way, movement of robot can be done simply by changing the value of cell as per requirement. The commands for running motion used in this implementation for one set of joints is written as

```
%No of token in particular place
      b1 = get_place('pLK_pos_al'); %Gets total tokens as place LK pos
      b2 = get_place('pLK_neg_al'); %Gets total tokens as place LK neg


            if eq(b1.tokens,25),
                %$Right Hand
                global_info.RE_angle = [5 0 0 2];
                global_info.RW_angle = [0 0 120 2];
                global_info.LK_angle = [-25 0 0 1];
                global_info.LA_angle = [0 0 -35 8];
            elseif eq(b2.tokens,25),
                global_info.RE_angle = [-30 0 0 2];
                global_info.RW_angle = [0 0 90 2];
                global_info.LK_angle = [25 0 0 1];
                global_info.LA_angle = [0 0 -25 8];

            end
```

During the running phase, joint representing Left Knee (LK) swings front and back making 25 degree in each direction. So, taking that as reference point, other joint's angle move along with it. When LK reaches 25 degree, it moves 25 degree in other direction and vice versa because running is a repetitive action and 25 is maximum degree it stretches while running. The total angle it has to cover is 50 degrees, for which it takes 50 simulation times. But for other joints it's different. Like for joint Right Elbow (RE), it has to cover only 35 degrees moving from -5 to 30. Since movement has to be done in time period of 50 simulation time. For this, firing has to be slower than regular firing hence its firing time is change to 2 for slow firing .After that this joint reaches its destination approximately at same time as that of joint LK (used as reference). Similarly, all joints are implementing in same way. The same process is carried out in other leg as well. Synchronization amongst the joints is achieved following this pattern. In this way, using Table 3-1 values, simulation of both running and walking of robot is achieved successfully.

## 4.6 Input file format for instruction

In this implementation, the moves of robot can be both stochastic and deterministic. The deterministic movement are coded inside robot's instructor and called when needed by simple single command. For stochastic movements, individual commands for motion for different joints are given separately at intended time. These commands are fed through a user defined file which is placed under the name '*instructFile.txt*', in the same folder where MSF resides. The structure of the file is shown below:

| Time | Task | Motor | Angle | Degree | Firing Time |
|---|---|---|---|---|---|
| 10 | 3 | LE | 1 | 5 | 0 |
| 30 | 3 | LE | 1 | 0 | 2 |
| 50 | 1 | 0 | 0 | 0 | 0 |
| 150 | 3 | LE | 1 | 15 | 0 |

**Table 4-1: Input format of instruction in a file**

Here, first column '*Time*' represent which time to fire, column '*Task*' indicates which task is to be performed like running, walking etc as defined, '*Motor*' corresponds to the joint, '*Angle*' is direction i.e. α is 1 , β is 2 and γ is 3. Other, '*Degree*' stands for value (degree) of movement of the joint and finally '*Firing Time*' indicates difference between each token to fire which can be viewed as one token per given firing time.

This file is read in MSF where all column values are seen as string at the beginning of the simulation. This is achieved by following code:

```
%For Inputing data from a file
fileID = fopen('myfile.txt','r');
global_info.inputData = textscan(fileID,'%s %s %s %s %s %s');
fclose(fileID);
```

As shown above, the file is read and all its values are stored in '*global_info.inputData'* as multi dimensional cell array. This array is passed on to the instructor. At first, the instructor reads the first row and extracts value of first column i.e. time. It waits until the time value obtained matches the current simulation time. As soon as the time matches, the instructor check the task to perform; this is situated in second row. If the task is 1 or 2 which is walking and running respectively, it checks no further values in the row and starts executing walking or running procedure which is already built in .Else if its task 3, it moves on and checks all the value in neighboring rows. This provides information about which motor (joint) to move in which direction (angle) along with movement degree value and firing time. After first row being read, it moves to another row and follows same process. More than one joint can be moved at same time by giving same time value. Note: the input commands must be sorted in ascending order in terms of time value.

## 4.7 Implementation in multiple robots (swarm robot)

After successful implementation of a single humanoid robot, the same functionalities are used to create multiple robots. The co ordinations between them are organized with the help of main instructor as shown in Figure 3-10. A single instructor feeds all robots with commands from instruction file. Each robot has their own instructor as well which contains the deterministic motions routines embedded inside it. So, the main instructor doesn't need to send all the steps rather only send the command that points to that routine. Modeling begins with creating Petri Net models which is defined in PDFs (Petri Net Definition Files) for different number of robots and their joints. The names of those PDFs are assigned in MSF as follows:

```
pns = pnstruct({...
                'instructor_pn_pdf',...
...             %Robot1
                'r1_instructor_pn_pdf',...
                'r1_head_pn_pdf','r1_trunk_pn_pdf',
                'r1_LeftElbow_pn_pdf','r1_RightElbow_pn_pdf',...
               'r1_leftWrist_pn_pdf','r1_rightWrist_pn_pdf',...
                'r1_LeftKnee_pn_pdf','r1_RightKnee_pn_pdf',...
                'r1_leftLegs_pn_pdf','r1_rightLegs_pn_pdf',...
...               %Robot2
               'r2_instructor_pn_pdf',...
                 'r2_head_pn_pdf','r2_trunk_pn_pdf',
                'r2_LeftElbow_pn_pdf','r2_RightElbow_pn_pdf',...
               'r2_leftWrist_pn_pdf','r2_rightWrist_pn_pdf',...
                'r2_LeftKnee_pn_pdf','r2_RightKnee_pn_pdf',...
                'r2_leftLegs_pn_pdf','r2_rightLegs_pn_pdf'});
```

Here only two robots are implemented. More robots can be created in same way. However tokens must be assigned separately to the Petri Net models similar as in one robot mode.

For instructions, the user defined file remains same but has one more column included i.e. '*robot*'. This column indicates for which the command was targeted. In this multi robot mode, a single main instructor controls the entire group of robots and maintains co ordination between them. The main instructor commands other instructors included in individual robots using following code where '*r1*' denotes robot1 and '*r2'* represents robot2.

```
global_info.r1_RK_angle =[20  0  0  0];
global_info.r1_LK_angle =[20  0  0  0];

global_info.r2_RK_angle =[20  0  0  0];
global_info.r2_lK_angle =[20  0  0  0];
```

The successful testing of simulation of single robot and synchronous act between groups of those robots in multi robot environment are displayed in form of simulation results in next section of the paper.

# 5 TEST AND RESULT

This implementation of Petri Net model of robot was coded and simulation was run on MATLAB environment using GPenSIM. First of all, a single mode robot was tested. The commands were given through an instruction file name ‘*instructFile.txt*’. The main simulation file is ‘*robot.m*’. The result of the simulation was displayed with the help of function ‘*plotpNew*’. This is a modified version of ‘*plotp*’ function available in MATLAB. Since this function was only able to distinguish between degrees of joints represented by tokens with different colors .This is not enough for situations where figures are viewed black and white. So, in this project, a different function was created with same functionalities but with an extra modification. With this new function ‘*plotpNew*’ not only colors but also different markers could be used to differentiate between tokens representing various joints and their movement directions. The modification that was made in original code and created a new function is shown below.

```
% For variable markers for diffrent lines in same graph
  markers = {'h','o','*','^','>','s','d','.','v','x','<','p','+'};

  for i = 1:numel(p), %p is output of plot
    set(p(i), 'Marker', markers{i});
  end;
```

The function ‘*plotpNew*’ was called as follows for different experiments to display token movements which represented joint movements in two different sets.

```
figure(1),plotpNew(sim,{ 'pLK_pos_al', 'pLK_neg_al','pLA_neg_ga', 'pRE_pos_al',
'pRE_neg_al', 'pRW_pos_ga'});

 figure(2),plotpNew(sim,{ 'pRK_pos_al', 'pRK_neg_al','pRA_neg_ga', 'pLE_pos_al',
'pLE_neg_al', 'pLW_pos_ga'});
```

## 5.1 Stochastic Motion (Single Robot)

For this experiment only task 3 was used which represents stochastic instruction. The main simulation file of GPenSIM was run and results shown in Figure 5-1 and Figure 5-2 were generated. This result was created with the command for stochastic movement given through the instruction file. The commands in the file are listed below in the Table 5-1.

| Time | Task | Motor | Angle | Degree | Firing Time |
|---|---|---|---|---|---|
| 10 | 3 | LE | 1 | 5 | 0 |
| 10 | 3 | LK | 1 | 5 | 0 |
| 30 | 3 | LE | 1 | 0 | 2 |
| 50 | 3 | RK | 1 | 10 | 0 |
| 50 | 3 | LK | 1 | -15 | 0 |
| 50 | 3 | RW | 3 | 20 | 0 |
| 70 | 3 | LE | 1 | 15 | 0 |
| 90 | 3 | RK | 1 | -5 | 2 |
| 95 | 3 | LA | 3 | -10 | 0 |

**Table 5-1: Input instruction for stochastic process**

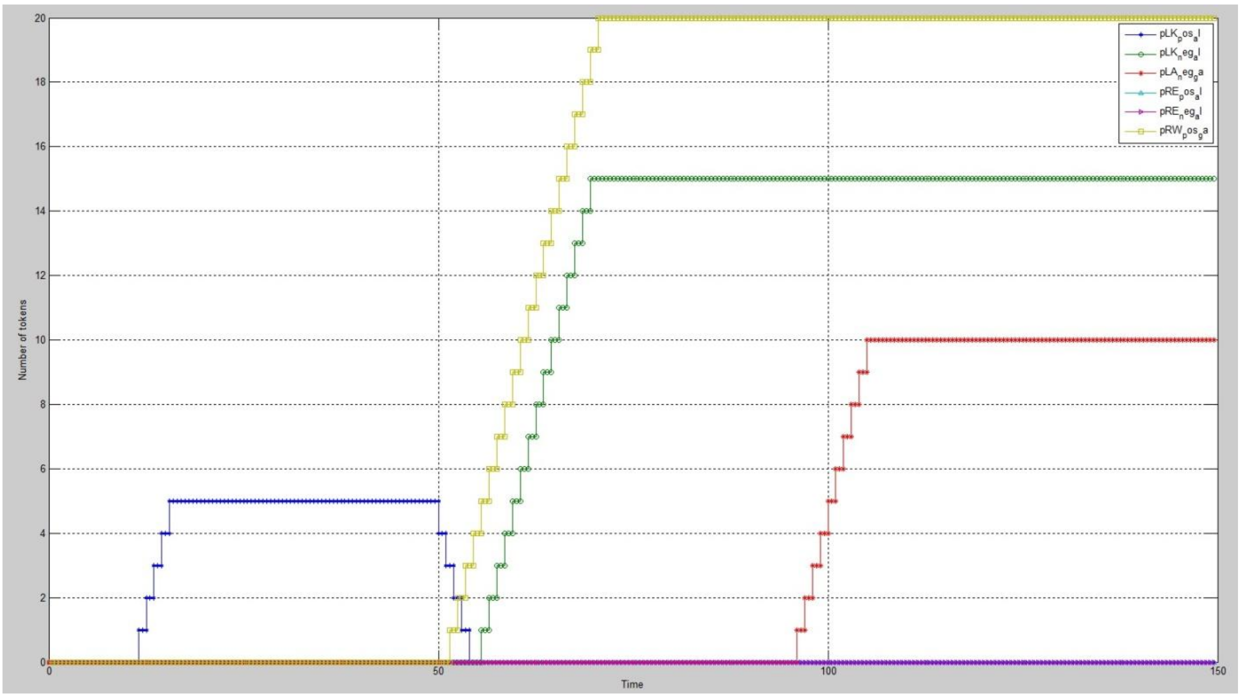


**Figure 5-1: Output of set I of joints (LK, LA, RE, RW) obtained from command set in; shown in Table 5-1.**

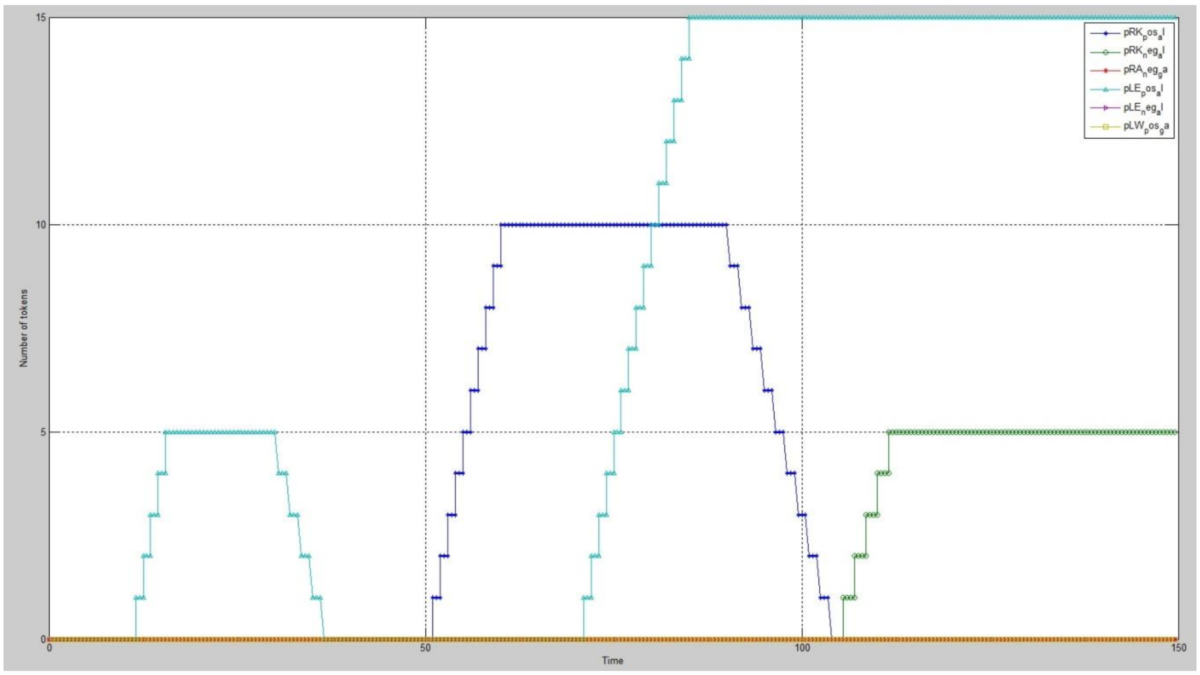


**Figure 5-2: Output of set II of joints (RK, RA, LE, and LW) obtained from command set in; shown in Table 5-1.**

In this way, set commands for the movement of the robot were executed. The user selected time for movements for any particular joints to degree it can move. Figure 5-1 and Figure 5-2 shows the output to the file whose values are set as shown in Table 5-1. Movements of joints LK , LA, RE and RW are shown in Figure 5-1 where as joints RK, RA, LE and LW are include in Figure 5-2. The only reason for displaying them in separate graph is to better visualize the flow of token (degree) of different joints. From above table, at time 10, robot is instructed to move joints 'LE' and 'LK' to 5 degree in positive direction. Each degree movement takes 1 simulation time. The graph is plotted between number of tokens and simulation time. As, discussed in design section, total number of token held in relevant places represent the degree of particular joint. In this first experiment only one angle of joint is considered as to minimize the complexity. Figure 5-1 can be translated to backend process where place '*pLK_pos_al*' begin to get one token per simulation time until it reaches 5 token at time 10. This procedure can be interpreted as **α** angle (i.e angle = 1) of joint LK is moved 5 degree at time 10. Similarly, in Figure 5-2, at time 10 the robot is instructed to move **α** angle of joint 'LE' represented by '*pLE_pos_al*' to move 5 degree as well. Since all joints are independent of each other, they can be commanded parallel irrespective of other joints. Moving on, at time 30 joint 'LE' moves back to 0 degree with firing time 2. As described before, firing time corresponds to slowness of the movement. In same way joints are moved according to other lines of command. One notable change is at time 50 where joint 'LK' is commanded to move -15 degrees. The current degree of 'LK' is 5. Moving from 5 to -15 should take 20 simulation times. First all tokens from place '*pLK_pos_al*' go to 0 and only after that place '*pLK_neg_al*' representing negative degree starts to receive tokens till it reaches 15 which corresponds to -15 degree movement of that particular joint. In this way, transitions from positive 5 to negative 15 takes place.

This result successfully represents a stochastic movement of a robot as commanded by the user. This increase and decrease of tokens represents movement of joint in particular angle with user commands.

## 5.2 Deterministic motion (Single Robot)

For Deterministic motion, only one line command is enough which determines the task to execute. Here are the results for walking and running simulations of robot presented in two separate figures corresponding to separate set of joints for each motion.

### 5.2.1 Walking motion

In case of walking, the values of joints are taken from Table 3-1 was generated by the gait cycle of a normal human. These values are hard coded inside the robot's instructor as discussed in implementation phase. So, only a simple task command can be send specifying the robot for walk motion. In this implementation it is '*task 1*'. So, input format is simple. Only time and task value are need to start the deterministic motion. This is shown below in Table 5-2 and it output is displayed underneath in Figure 5-3 and Figure 5-4 as different sets of joints.

| Time | Task | Motor | Angle | Degree | Firing Time |
|---|---|---|---|---|---|
| 10 | 1 | 0 | 0 | 0 | 0 |

**Table 5-2: Input instruction for deterministic process of walking**

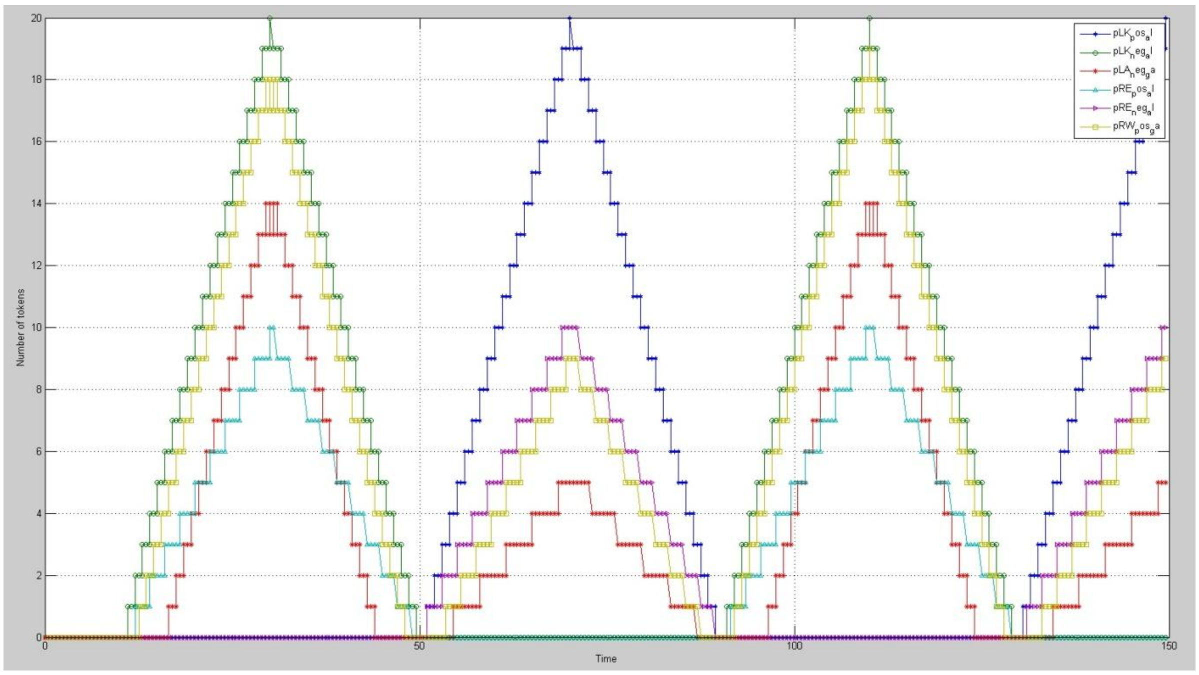


**Figure 5-3: Output of set I of joints (LK, LA, RE, RW) obtained from command set in; shown in Table 5-2.**

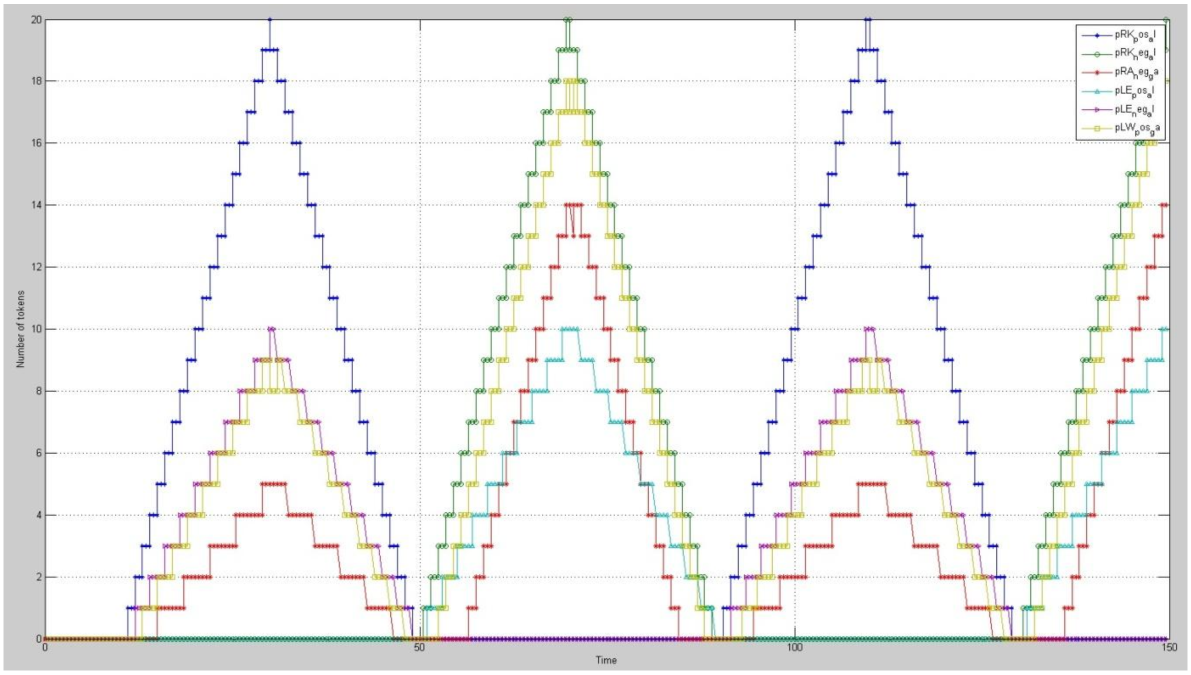


**Figure 5-4: Output of set II of joints (RK, RA, LE, and LW) obtained from command set in; shown in Table 5-2.**

Figure 5-3 and Figure 5-4 shows movement of joints for walk motion. These are repetitive motions. The first figure contain joints ‘LK’ , ‘LA’, ‘RW’, and ‘RE’ as joint set 1 where are second figure contains ‘RK’, ‘RA’, ‘LW’ and ‘LE’ as joint set 2. The motion start from time 10 as specified in the command line. When one set is in forward motion other is in opposite direction as specified by gait cycle. This is how walking motion takes place. This result shows exactly as expected with joints of legs along with hand swings. Some joints take longer time than other to move. But all joints of each set reaches to its destination at same time as specified by the gait cycle. Hand swings less than leg movement while walking so, hand joints move slow in comparison to legs joints. This is implemented by using variable firing times. With help of it, the walking simulation is successfully achieved.

### 5.2.2 Running Motion

The running simulation is exactly similar to walking in terms of implementation. The only differences are the command value input and gait cycle used. The command is stated to instruct by ‘*task 2*’. The movements of joints are already embedded inside instructor as that of walking. Following file format is used shown in Table 5-3 and generated outputs are shown in Figure 5-5 and Figure 5-6.

| Time | Task | Motor | Angle | Degree | Firing Time |
|---|---|---|---|---|---|
| 10 | 2 | 0 | 0 | 0 | 0 |

**Table 5-3: Input instruction for deterministic process of running**

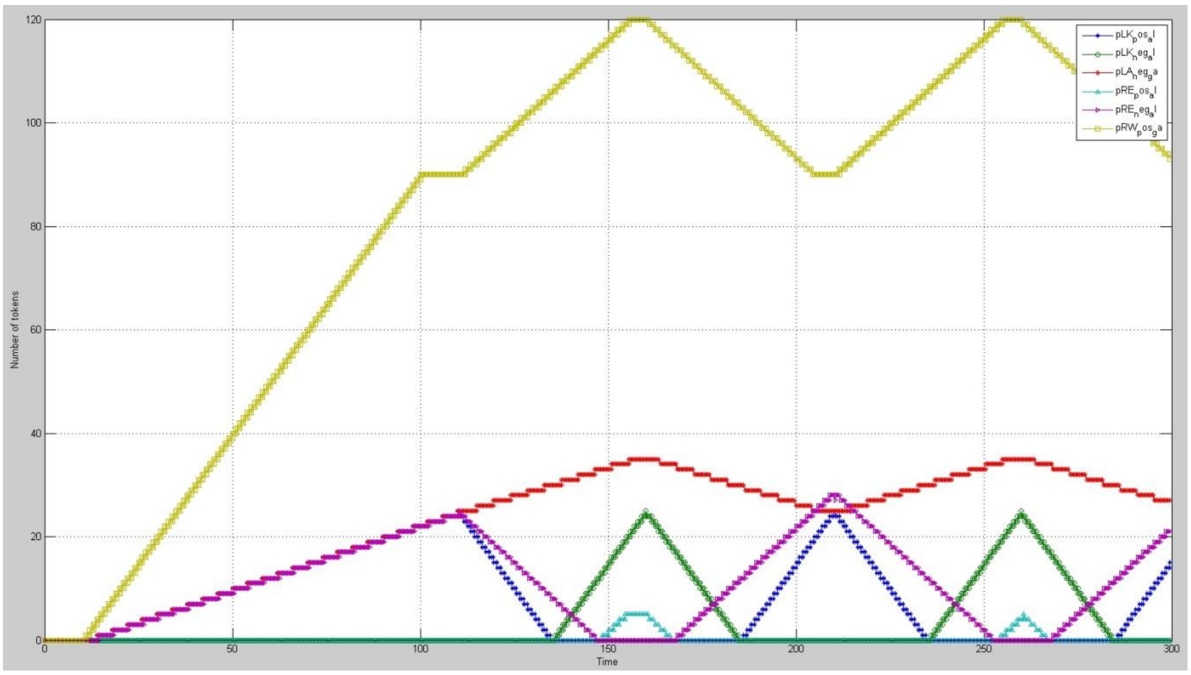


**Figure 5-5: Output of set I of joints (LK, LA, RE, RW) obtained from command set in; shown in Table 5-3.**

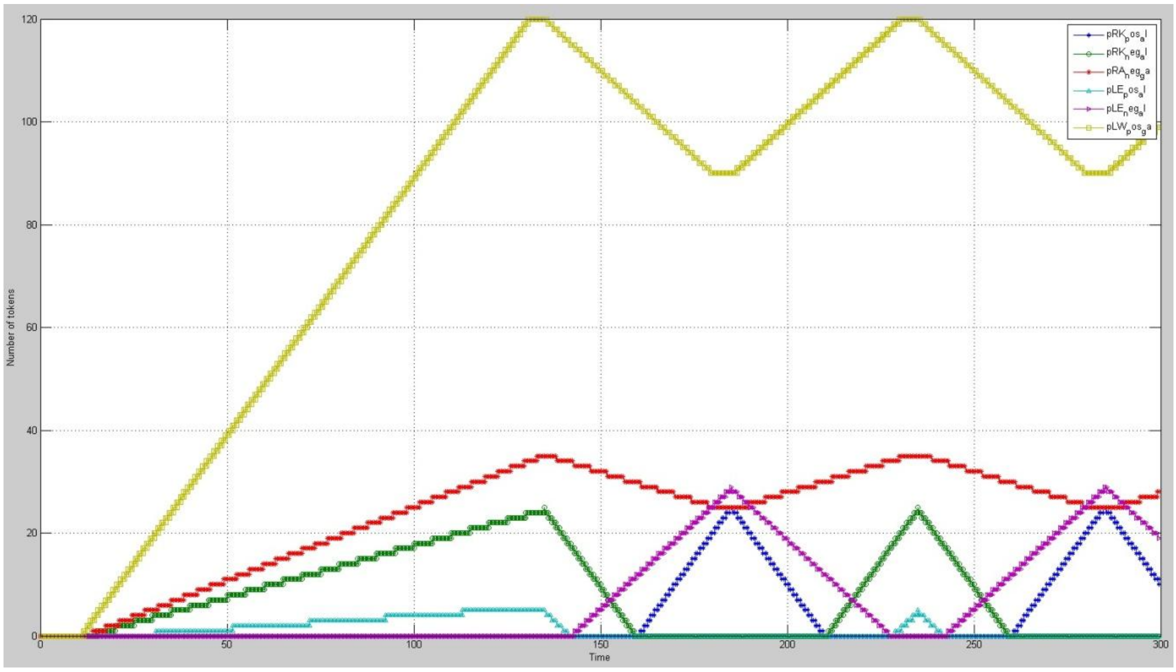


**Figure 5-6: Output of set II of joints (RK, RA, LE, and LW) obtained from command set in; shown in Table 5-3.**

Similar to walking motion, here in running motion, when one set of joints are in forward motion; other set has to be in opposite side. This is taking place in the simulation result shown in both figures. The degree values are taken from same gait cycle and implemented accordingly.

## 5.3 Combined stochastic and deterministic motions (Single Robot)

Both stochastic and deterministic motions can also be implemented in single simulation through same instruction file. As discussed many time in this paper, task 1 and 2 are for deterministic motion and task 3 represents stochastic motion. The simulation combining both can be seen in a simulation using instruction file below.

| Time | Task | Motor | Angle | Degree | Firing Time |
|---|---|---|---|---|---|
| 10 | 3 | LE | 1 | 5 | 0 |
| 15 | 3 | RE | 1 | 10 | 0 |
| 30 | 3 | LE | 1 | 0 | 2 |
| 30 | 3 | RE | 1 | 0 | 0 |
| 50 | 1 | 0 | 0 | 0 | 0 |
| 150 | 3 | LE | 1 | 15 | 0 |

**Table 5-4: Input instruction for combination of both stochastic and deterministic process**

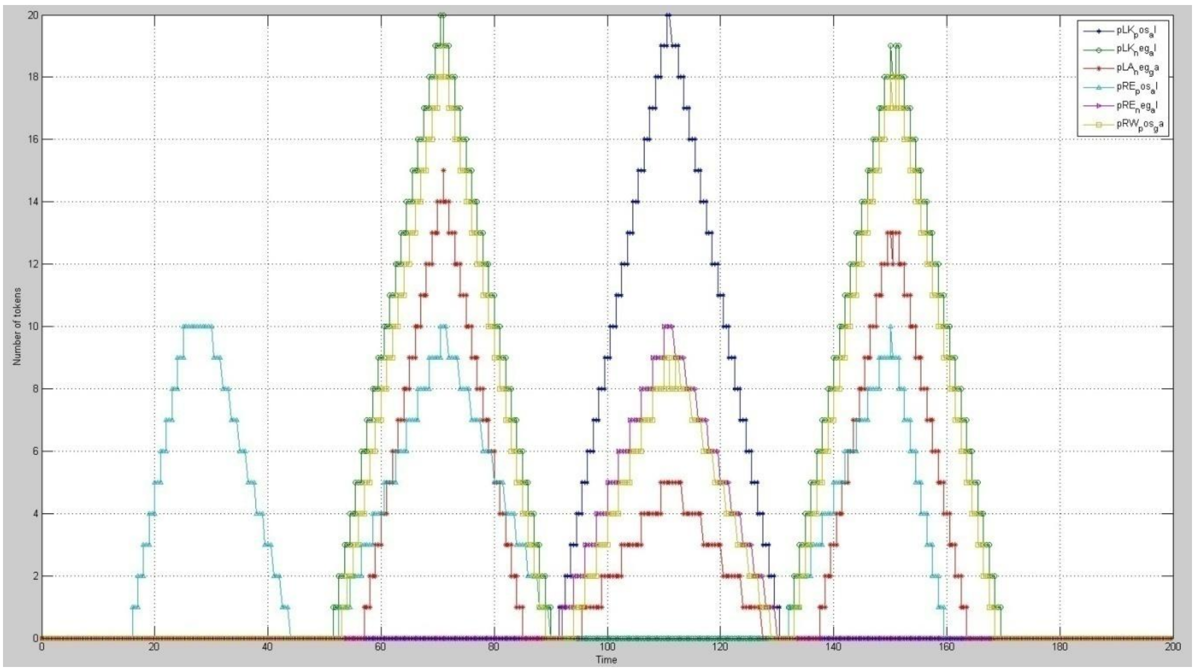


**Figure 5-7: Output of set I of joints (LK, LA, RE, RW) obtained from command set in; shown in Table 5-4.**

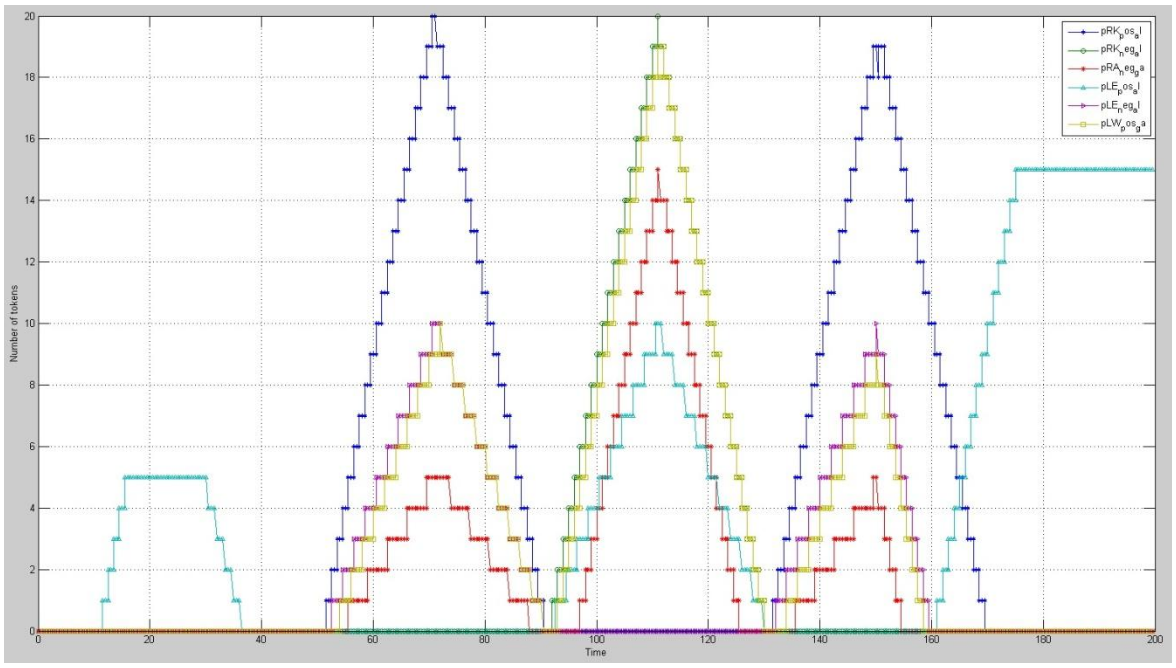


**Figure 5-8: Output of set II of joints (RK, RA, LE, LW) obtained from command set in; shown in Table 5-4.**

In time 10, stochastic motion denoted by '*task 3*' is set as command where joint 'LE' is ordered to movie 5 degrees and again in time 30, same joint called to moved back to 0 degree. On third step, task 1 is called denoted as walking motion in time 50. The robot begins the walk motion. At time 150, deterministic motion is interrupted by stochastic motion which commands to move joint 'LE' to 15 degrees. In this way both motion can be used in same simulation.

## 5.4 Multi (Swarm) robots environment

After successful simulation of single robots and as described in the implementation section, same models were used for multiple robot environments. For this different robots have their own instructor. On top of that there is one main instructor for synchronization between the robots movements. These robots are totally independent with each other same as their joints. For this experiment only two robots are used for better explanation and less complexity. Addition of more robots is not difficult and can be easily achieved. The same instruction structure used in single robot environment is put into action here as well with a new column '*robot*'. This column specifies which robot this dispatched command is for. All robots can be commanded parallel. One robot can be going through repetitive (deterministic) motion like walking or running and at the same time other can be doing stochastic motion or both can be moving same joints as a dance routine or so on. For this experiment, some of the instructions are input as shown in Table 5-5. Their output results are displayed in Figure 5-9 and Figure 5-10 for robot 1 and Figure 5-11 and Figure 5-12 for robot 2.

Instructions to the instructor for multiple robot environments is shown below in table.

| **Time** | **Robot** | **Task** | **Motor** | **Angle** | **Degree** | **Firing Time** |
|---|---|---|---|---|---|---|
| 10 | 1 | 3 | LE | 1 | 5 | 0 |
| 10 | 1 | 3 | LK | 1 | 5 | 0 |
| 10 | 2 | 3 | LE | 1 | 5 | 0 |
| 10 | 2 | 3 | LK | 1 | 5 | 0 |
| 30 | 1 | 3 | LE | 1 | 0 | 2 |
| 30 | 2 | 3 | LE | 1 | 0 | 2 |
| 50 | 1 | 3 | RK | 1 | 10 | 0 |
| 50 | 1 | 3 | LK | 1 | -15 | 0 |
| 50 | 1 | 3 | RW | 3 | 20 | 0 |
| 50 | 2 | 3 | RK | 1 | 10 | 0 |
| 50 | 2 | 3 | LK | 1 | -15 | 0 |
| 50 | 2 | 3 | RW | 3 | 20 | 0 |
| 70 | 1 | 3 | LE | 1 | 15 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 70 | 2 | 3 | LE | 1 | 15 | 0 |
| 90 | 1 | 3 | RK | 1 | -5 | 2 |
| 90 | 2 | 3 | RK | 1 | -5 | 2 |
| 95 | 1 | 3 | LA | 3 | -10 | 0 |
| 95 | 2 | 3 | LA | 3 | -10 | 0 |
| 120 | 1 | 1 | 0 | 0 | 0 | 0 |
| 120 | 2 | 2 | 0 | 0 | 0 | 0 |

**Table 5-5: Input instruction for combination of both stochastic and deterministic process in swarm robot environment.**

## Robot 1

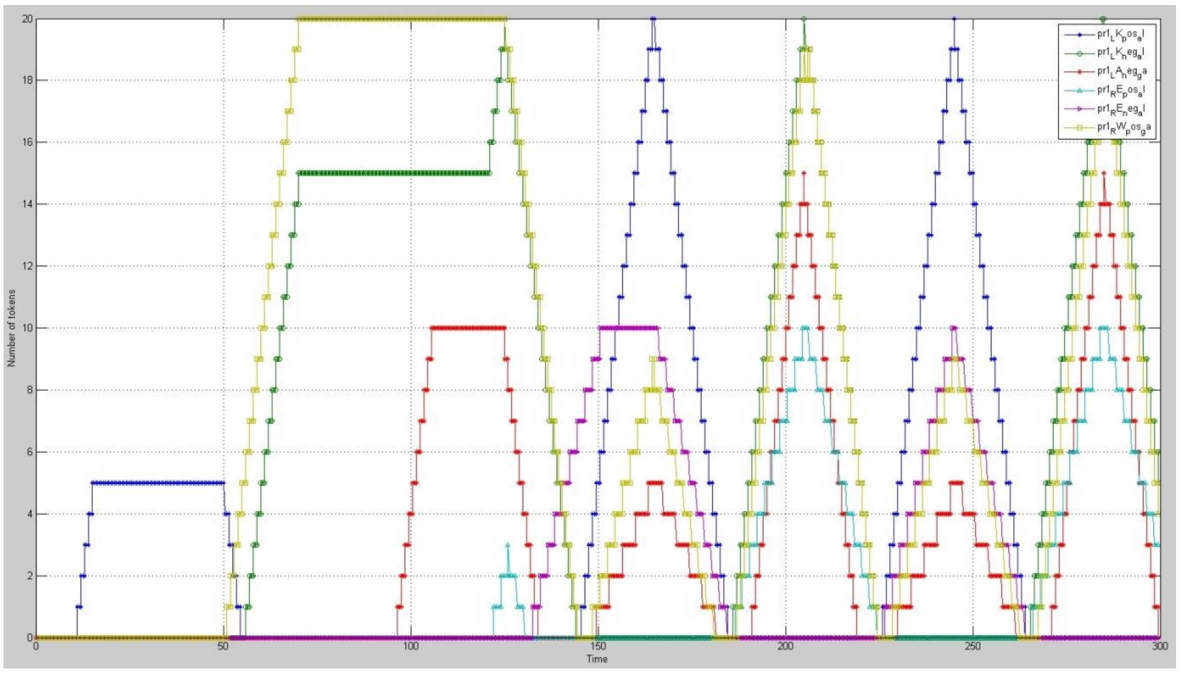


**Figure 5-9: Output of set I of joints (LK, LA, RE, RW) of robot 1, obtained from command set in; shown in Table 5-5.**

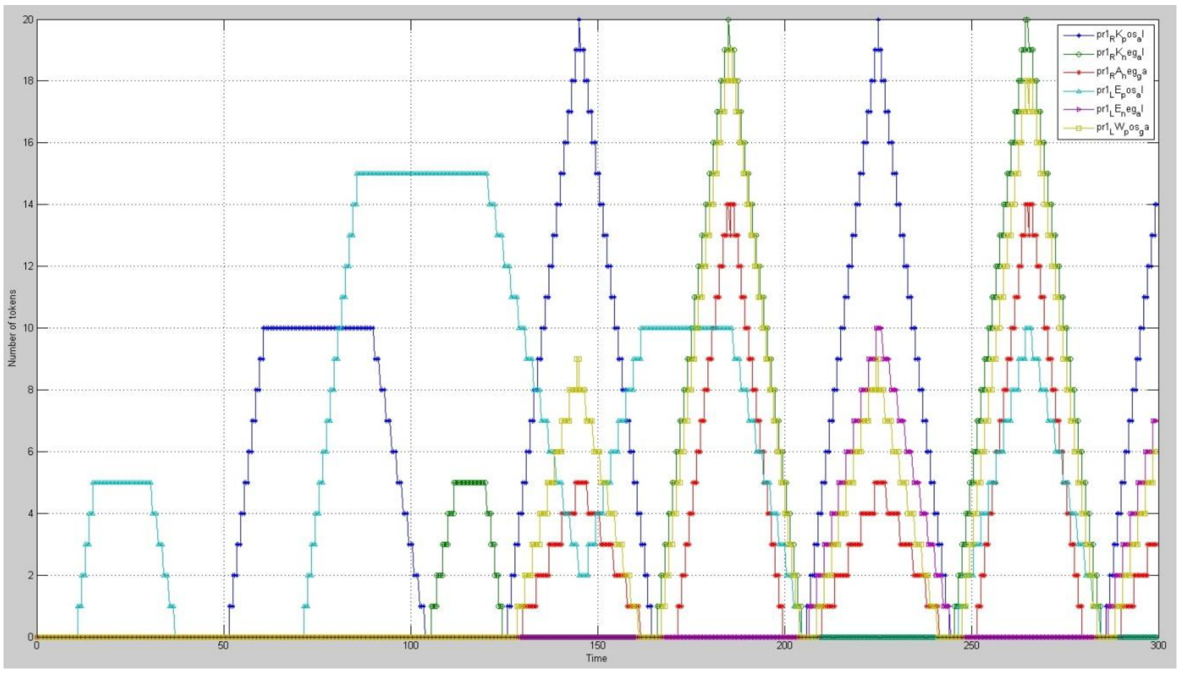


**Figure 5-10: Output of set II of joints (RK, RA, LE, LW) of robot1, obtained from command set in; shown in Table 5-5.**

## Robot 2

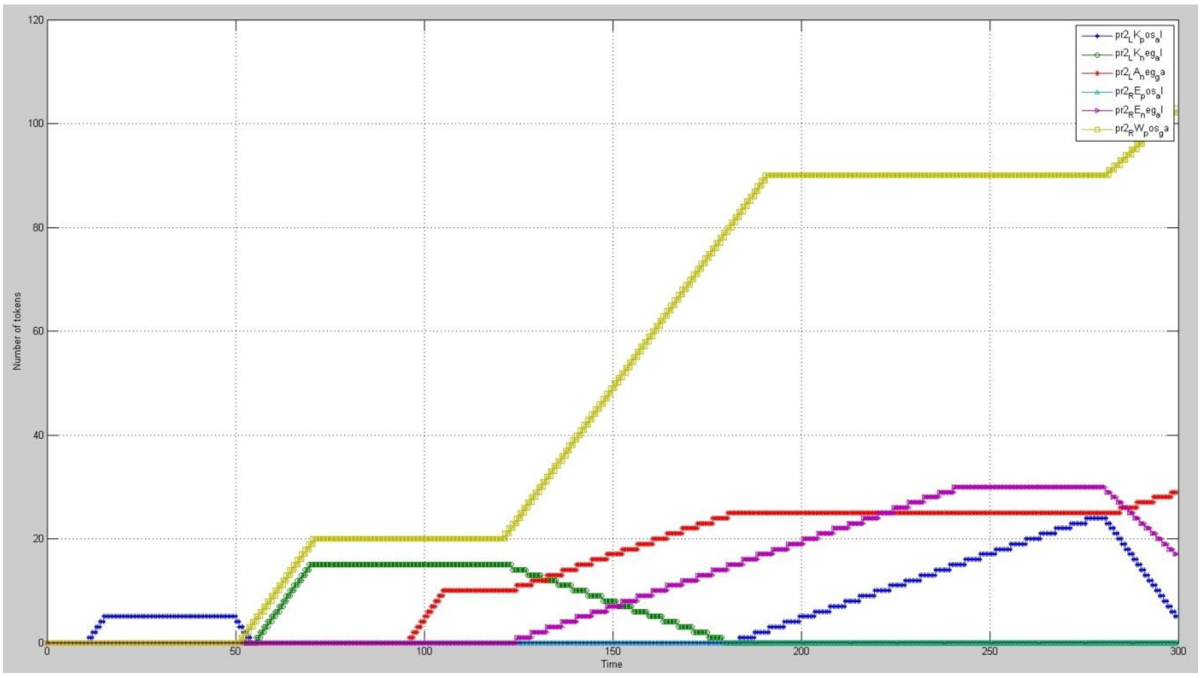


**Figure 5-11: Output of set I of joints (LK, LA, RE, RW) of robot 2, obtained from command set in; shown in Table 5-5.**

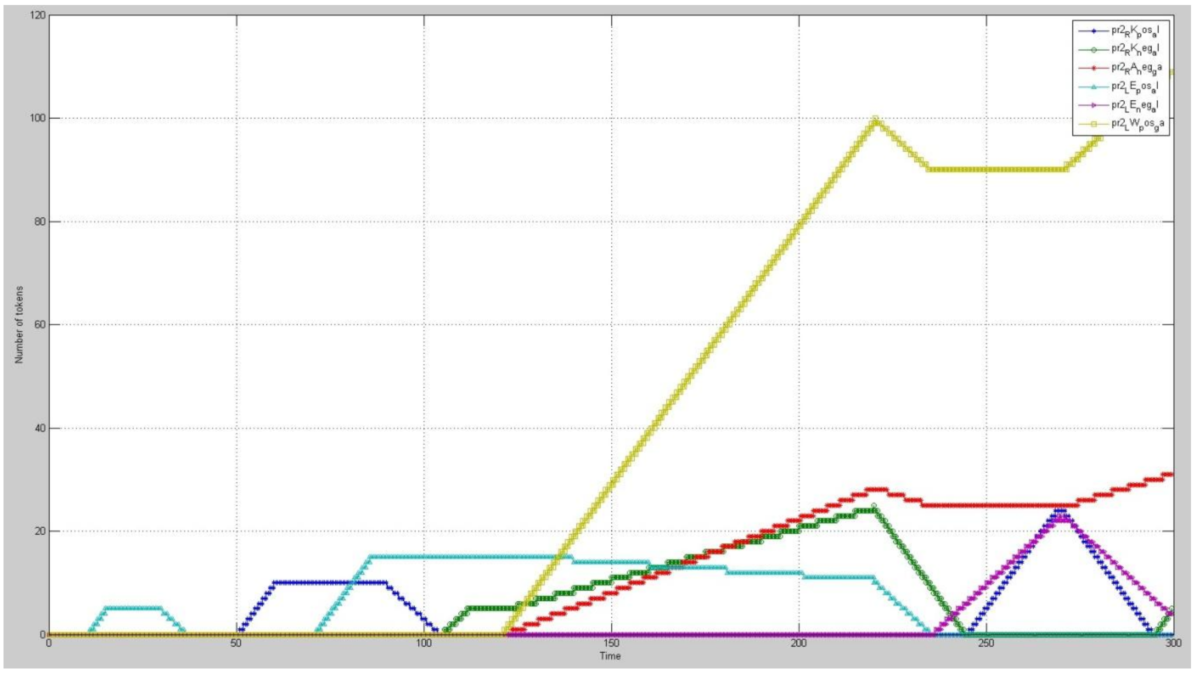


**Figure 5-12: Output of set II of joints (RK, RA, LE, LW) of robot 2, obtained from command set in; shown in Table 5-5.**

These figures mark the successful implementation of the model in multiple robot environments as well. Similar to previous division of set of joints, here too joints are put in separate sets. The reason was same as before for better representation of movements of joints. Figure 5-9 and Figure 5-11 displays movements of token of joint ‘RE’, ‘RW’, ‘LK’ and ‘LA’ of robot1 and robot 2 where as Figure 5-10 and Figure 5-12 represents joints ‘LE’, ‘LW’, ‘RK’ and ‘RA’ of ‘*robot 1*’ and ‘*robot 2*’ respectively. The movements take place according to the instruction in the file. Both robots make similar stochastic movements defined by user till time 95. In time 120, ‘*robot 1*’ starts walking motion and ‘*robot 2*’ begins to run for remaining simulation time. A single supervisor controlled two robots through a single set of instruction from a file. These commands create synchronization between them by making same movements and later ending with different motions that is walk and run for robot 1 and robot 2 respectively.

Chapter 6

# 6 DISCUSSION AND CONCLUSION

This paper comprises of humanoid robot whose actions and parts are visualized in 3-dimensional environment. According to [33],there are seven approaches to simulate human body: kinematics, dynamics, motion controllers, animation of deformable objects, motion planning, and autonomous character behavior and optimization methods. Amongst these approaches, the most appropriate one was chosen: kinematics. In this approach, to generate human motion in DES was efficient and compatible in terms of our necessity. In kinematics, human bodies are defined as a system which is presented by the position and orientation of joints. These directly relate to discrete events. There are two types; forward and inverse kinematics. Even though inverse kinematics provides more general and realistic motions, forward kinematics is used for its simple computation and time requirements for finding new position. The robots were modeled and simulated with reference to the paper describing motion of human as Discrete Event System (DES) of a real time collaboration that follows forward kinematics for its deterministic and stochastic motions. The main focus in this paper was not given to how realistic the robot looks rather to test the human model and simulate in Petri Net. Petri Net is a powerful tool providing concurrency, parallelism and synchronization [24] for dynamic systems. GPenSIM is capable of simulating DES perfectly using Petri Net tools. Furthermore, it provides flexibility, extensibility and ease of use [2]. But there are some short comings with this version of GPenSIM. The most prominent which makes big impact is that it doesn't incorporate online simulator. This makes it impossible to monitor and control the state of system during simulation run. Simulation parameters are input at the beginning of simulation and results are presented at the end of it.

While replicating a human deterministic motion, there were some difficulties. The joints that make up any particular motion doesn't necessary move at the same velocity. They may have same starting time and ending time but may have different degree to cover. This is mostly covered by variable firing time in GPenSIM. But the issue is, same joint movement can be quick or slow depending upon the task to perform. Current version of GPenSIM is not capable of handling this problem. The firing time for different transition can only be varied at the starting of simulation run. No dynamic changes during simulation are possible. In this project, this problem is handled using blocking mechanism in pre processors which is a Transition definition file (TDF). Since the mechanism is implemented on the top level, output value altered with changes in delta time (sampling time) of the simulation.

## 6.1 Future work

To create a robot with more accuracy and validity, a real world human ROM was considered. The accuracy can’t be fully proven until it is implemented in a real robot. GPenSIM supports [28] interacting with external devices with the use of TDF (pre and post processors). There are various discrete control applications developed by using GPenSIM over several toolboxes. Most common experiments were done on LEGO robots which can be controlled using PC through a USB connection or wireless Bluetooth connections. In future, the comparison and validation can take place by implementing this paper’s work on a real physical robot.

More realistic motion can be generated by deploying more complex methods in DES with help of real- time reproducible inverse kinematics techniques [41]. Even fast inverse kinematics for human motion represented as DES can be considered. With use of inverse kinematics, coordinates for next motion can be generated randomly and new path from current location to new location can be estimated giving more realistic human motion. Though this requires intense calculation in comparison to forward kinematics, it deserves some research. This will help to create a self controlling and more realistic robot.

## 6.2 CONCLUSION

A humanoid robot was successfully modeled and simulated in Petri Net which was coded in MATLAB with general purpose Petri Net Simulation tool also known as GPenSIM. A humanoid model in this paper was viewed has discrete event system where events refers to as the motion of sixteen limbs that could be simulated independently and concurrently. Furthermore, forward kinematics governing the motions of those limbs were viewed as a rigid body in 3D space. The Range of model (ROM) of a healthy human was implied to restrict movements of those limbs. Stochastic and deterministic motions of human body were simulated to its implementation. For deterministic motion, human gait cycle for walking and running were discussed and provided as input for the robot. This model was later implemented in the multiple robot environments with same design. Everything worked as anticipated. But for better co ordinations between robots, a single main supervisor was added which instructed each robot. A single coordinator handling communication over all robots would have had problem due to overloading of communication channel due to passing on all steps of instructions. To cope with this problem, each robot was build with its own instructor as well; which contained most of the common routines like steps for running, walking and so on and communicated directly to main instructor. In this way, main instructor didn’t have to send all the steps of requested routine to perform. Instead it had to only send a simple command like '*task 1*' to '*robot 1*' which would mean run, walk etc. The task and related routine have to be predetermined by all the robots and instructor before simulation begins.

And hence a Petri net model of a single humanoid robot was successfully modeled and extended to multiple (group) coordination and these swarm of robots were able to perform coordinated tasked controlled by a single supervisor.

# REFERENCES

[1] M. Panggabean, L. A. Rønningen, and H. Øverby, "Modeling and simulating motions of human bodies in a futuristic distributed tele-immersive collaboration system for synthesizing transient input traffic," *Simulation Modelling Practice and Theory*, vol. 31, pp. 132-148, 2013.

[2] R. Davidrajuh, "GPenSim: A New Petr NEt Simulator," ed. http://www.davidrajuh.net/gpensim, 2010.

[3] G. Galileo educational Network. (10 May). *Introduction To Robots*. Available: http://www.galileo.org/robotics/intro.html

[4] M. Dorigo, M. Birattari, and M. Brambilla, "Swarm robotics," *Scholarpedia*, vol. 9, p. 1463, 2014.

[5] SUPSI. (2015). *Humanoid Robotic Swarms*. Available: http://www.supsi.ch/home_en/ricerca/progetti/inevidenza/Swarmanoid1.html

[6] J. Wang, "Petri nets for dynamic event-driven system modeling," *Handbook of Dynamic System Modeling*, pp. 1-17, 2007.

[7] D. Angeli, P. De Leenheer, and E. D. Sontag, "A Petri net approach to the study of persistence in chemical reaction networks," *Mathematical Biosciences*, vol. 210, pp. 598-618, 2007.

[8] L. Szilágyi, "Random Process Simulation Using Petri Nets," *MACRo 2015*, vol. 1, pp. 177-182, 2015.

[9] L. Fossati and D. Varacca, "A petri net model of handshake protocols," *Electronic Notes in Theoretical Computer Science*, vol. 229, pp. 59-76, 2009.

[10] Y. Lv, C. Lee, Z. Wu, H. Chan, and W. Ip, "Priority-based distributed manufacturing process modeling via hierarchical timed color Petri net," *Industrial Informatics, IEEE Transactions on*, vol. 9, pp. 1836-1846, 2013.

[11] T. M. Chen, J. C. Sanchez-Aarnoutse, and J. Buford, "Petri net modeling of cyber-physical attacks on smart grid," *Smart Grid, IEEE Transactions on*, vol. 2, pp. 741-749, 2011.

[12] M. P. Fanti, G. Iacobellis, A. M. Mangini, and W. Ukovich, "Freeway traffic modeling and control in a first-order hybrid petri net framework," *Automation Science and Engineering, IEEE Transactions on*, vol. 11, pp. 90-102, 2014.

[13] M. Araújo and L. Roque, "Modeling games with petri nets," *Breaking New Ground: Innovation in Games, Play, Practice and Theory. DIGRA2009. Londres, Royaume Uni*, 2009.

[14] M. A. Syufagi, M. Hariadi, and M. H. Purnomo, "Petri net model for serious games based on motivation behavior classification," *International Journal of Computer Games Technology*, vol. 2013, p. 1, 2013.

[15] F. Ahmad, I. Fakhir, S. A. Khan, and Y. D. Khan, "Petri net-based modeling and control of the multi-elevator systems," *Neural Computing and Applications*, vol. 24, pp. 1601-1612, 2014.

[16] O. Fukuda, J. Kim, I. Nakai, and Y. Ichikawa, "EMG control of a pneumatic 5-fingered hand using a Petri net," *Artificial Life and Robotics*, vol. 16, pp. 90-93, 2011.

[17] F. Basile, C. Carbone, and P. Chiacchio, "Simulation and analysis of discrete-event control systems based on Petri nets using PNetLab," *Control Engineering Practice,* vol. 15, pp. 241-259, 2007.

[18] C. Su and H. Li, "An affective learning agent with Petri-net-based implementation," *Applied Intelligence,* vol. 37, pp. 569-585, 2012.

[19] H. Costelha and P. Lima, "Robotic Tasks Modeling and Analysis Based on Petri Nets."

[20] R. G. Ingalls, "Introduction to simulation," in *Proceedings of the 40th Conference on Winter Simulation*, 2008, pp. 17-26.

[21] A. Maria, "Introduction to modeling and simulation," in *Proceedings of the 29th conference on Winter simulation*, 1997, pp. 7-13.

[22] J. Banks, J. S. Carson, and B. L. Nelson, *Discrete-event System Simulation*: Prentice Hall, 1996.

[23] J. E. Reeb and S. A. Leavengood, "Simulating a manufacturing system: an introduction," Corvallis, Or.: Extension Service, Oregon State University2003.

[24] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE,* vol. 77, pp. 541-580, 1989.

[25] M. Barad, "Tutorial and Survey Articles: An introduction to Petri Nets," *International Journal of General Systems,* vol. 32, pp. 565-582, 2003.

[26] J. L. Peterson, "Petri net theory and the modeling of systems," 1981.

[27] R. Davidrajuh, "Extended P-N models," in *Discrete Simulation and Performance Analysis*, ed. class notes of DAT530 at UiS, 2014.

[28] R. Davidrajuh, "GPenSIM," in *A general purpose Petri net simulator for mathematical modeling and simulation of discrete-event systems in MATLAB*, ed. http://www.davidrajuh.net/gpensim, 2015.

[29] K. Jensen, "Coloured Petri nets and the invariant-method," *Theoretical computer science,* vol. 14, pp. 317-336, 1981.

[30] M. Panggabean, "Towards Geographically-Distributed Immersive Collaborations with Delay Guarantee: Modeling, Simulation, Synthesis, and Compression," 2014.

[31] C.-F. Kuo and M.-J. J. Wang, "Motion generation and virtual simulation in a digital environment," *International Journal of Production Research,* vol. 50, pp. 6519-6529, 2012.

[32] P. Ramamurthy, M. V. Blundell, C. Bastien, and Y. Zhang, "Computer simulation of real-world vehicle–pedestrian impacts," *International Journal of Crashworthiness,* vol. 16, pp. 351-363, 2011.

[33] A. I. Zhmakin, "Mathematics of human motion: from animation towards simulation (a view form the outside)," *arXiv preprint arXiv:1102.4992,* 2011.

[34] J. P. Granieri, J. Crabtree, and N. I. Badler, "Production and playback of human figure motion for visual simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS),* vol. 5, pp. 222-241, 1995.

[35] H. Hatze, "A mathematical model for the computational determination of parameter values of anthropomorphic segments," *Journal of Biomechanics*, vol. 13, pp. 833-843, 1980.

[36] A. Faller, M. Schünke, G. Schünke, and E. Taub, *The human body: an introduction to structure and function*: Thieme, 2004.

[37] N. A. a. S. A. (NASA). (1995). *Anthropometry and biomechanics (Vol. 1, Section 3) inMan-Systems Integration Standards (NASA-STD-3000) Revision B*. Available: http://msis.jsc.nasa.gov/sections/section03.htm

[38] R. N. Jazar, *Theory of applied robotics: kinematics, dynamics, and control*: Springer Science & Business Media, 2010.

[39] T. F. Novacheck, "The biomechanics of running," *Gait & posture*, vol. 7, pp. 77-95, 1998.

[40] J. R. Franz, K. W. Paylo, J. Dicharry, P. O. Riley, and D. C. Kerrigan, "Changes in the coordination of hip and pelvis kinematics with mode of locomotion," *Gait & posture*, vol. 29, pp. 494-498, 2009.

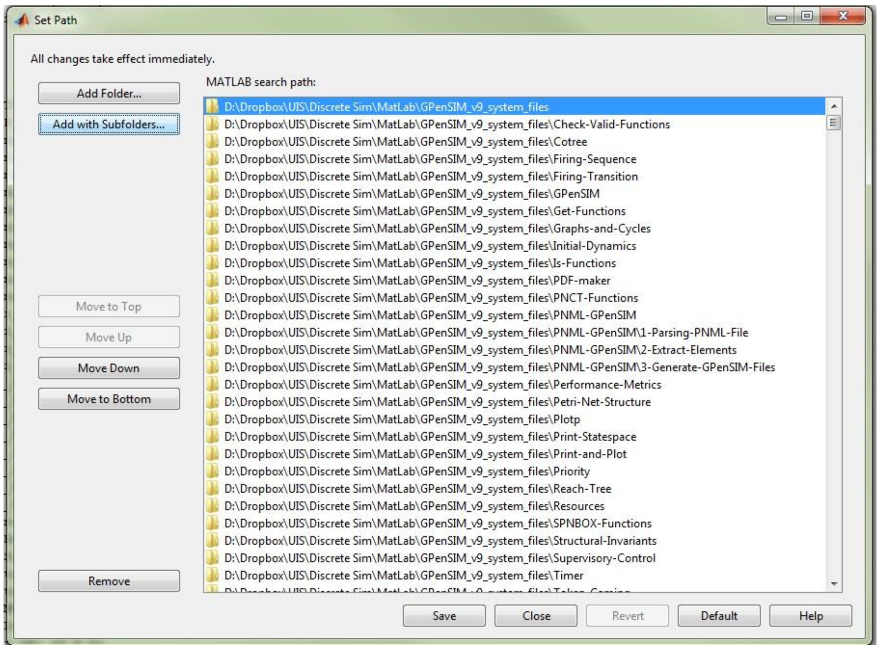
[41] D. Tolani, A. Goswami, and N. I. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs," *Graphical models*, vol. 62, pp. 353-388, 2000.

# Appendix A

Installing GPenSIM and configuring working directory,

1. First of all, download 'GPenSIM' from the author's website. This is available in zipped file format in website: "http://www.davidrajuh.net/gpensim/".
2. The downloaded file should be unzipped and placed somewhere in hard drive. All the files necessary for operating GPenSIM are included inside the unzipped folder.
3. GPenSIM is written in MATLAB as an advanced tool to model and simulate large and complex system. So, it operates within the MATLAB environment. To integrate GPenSIM, a path pointing toward the unzipped folder must be set. This can be done in following ways:

- Open MATLAB, Go to file menu and select "set path" command
- A new window will pop up under the window name "Set Path", press "Add with subfolder" which includes all the folders inside the unzipped folder. This is necessary because the unzipped folder contains variable modules of codes inside different folder for various functions.
- Select the directory and save it.



All changes take effect immediately.

MATLAB search path:

```
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Check-Valid-Functions
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Cotree
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Firing-Sequence
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Firing-Transition
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\GPenSIM
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Get-Functions
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Graphs-and-Cycles
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Initial-Dynamics
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Is-Functions
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\PDF-maker
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\PNCT-Functions
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\PNML-GPenSIM
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\PNML-GPenSIM\1-Parsing-PNML-File
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\PNML-GPenSIM\2-Extract-Elements
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\PNML-GPenSIM\3-Generate-GPenSIM-Files
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Performance-Metrics
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Petri-Net-Structure
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Plotp
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Print-Statespace
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Print-and-Plot
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Priority
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Reach-Tree
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Resources
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\SPNBOX-Functions
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Structural-Invariants
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Supervisory-Control
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Timer
D:\Dropbox\UIS\Discrete Sim\MatLab\GPenSIM_v9_system_files\Token-Coming
```

- To test if the GPenSIM is set up correctly. Go to MATLAB command window and type '*gpensim*' and press enter, if result similar to output below is displayed, then everything is working normally. The installation is complete. All the implementations were tested in version 8 as shown below.

```
>> gpensim
--------
GPenSIM version 8.0;   Lastupdate: August 2013
 (C) Reggie.Davidrajuh@uis.no
 ttp://www.davidrajuh.net/gpensim
--------
```

4. After GPenSIM is set up, similar process must be followed to include all folders and subfolder containing code for complete modeling and simulation of humanoid robot. All codes are separated in different files as per GPenSIM formatting. The folder and subfolder containing these files are zipped into one file, named '*Modeling_with_petrinets(Master Thesis 2015).zip*'.
5. Extract the zipped file into a directory and set the path to that directory along with all subfolders following same procedure used to set up GPenSIM.

# Appendix B

User manual for beginning simulation of humanoid robot is as follows:

1. There are two different sections for simulating robots as discussed in implementation section of this paper. The first one is a single robot and second is multi robot. These are implemented separated and stored differently in two different folders inside main folder '*Modeling_with_petrinets(Master Thesis 2015)'*.
2. For single robot mode, all necessary files are included in subfolder named '*petriNet(Robot)*' inside main folder. The instructions for movement of robot joints should be entered through a text file inside the sub folder named '*instructFile.txt*'. The input format is explained in implementation section.
3. For multi robot mode, similar to single mode, all files are included in separated sub folder. The name given to multi mode sub folder is '*petriNet(Robot)_swarm*'. Instructions should be given in same fashion as single mode and are entered through different file but same name '*instructFile.txt*' which is placed inside the subfolder of multimode robot.
4. After instructions are input and saved. Simulation can begin by typing '*robot*' for single mode or '*robot_swarm'* for multi robot mode in the command window of MATLAB.
5. The results are displayed in a graphical format drawn against token **VS** time. Here, a token movement corresponds to joint's degree movement is displayed, exactly similar to results displayed in '*Test and Result*' section of the paper.

# Appendix C

## C1. Source code of single robot mode

### 1. The main simulations file (MSF)

**robot.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The MSF for single robot mode
% Last modified: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all; clc;

global global_info;
global_info.DELTA_TIME= 0.5; %timer increment value is 0.5 sec(high sampling rate)
global_info.STOP_AT = 200; %stop simulation after 200 time


 %To initialize all angles values to zero
  initialize();


%counter used by instructor, begins from second line to avoid the title
%first line in instruction file
global_info.counter = 2;

%For Inputing data from a file : 'instructFile.txt'
fileID = fopen('instructFile.txt','r');
global_info.inputData = textscan(fileID,'%s %s %s %s %s %s');
fclose(fileID);

 %obtain row size of an array
 global_info.rowSize=cellfun(@length,global_info.inputData);


global_info.task = 3;  %Variable indicating diffent task
global_info.pre_task = 3;  %To store previous task performed




%Passing of PDFs to function 'pnstruct'
pns = pnstruct({...
                'instructor_pn_pdf',...
...%                 'head_pn_pdf','trunk_pn_pdf',
                'LeftElbow_pn_pdf','RightElbow_pn_pdf',...
               'leftWrist_pn_pdf','rightWrist_pn_pdf',...
                'LeftKnee_pn_pdf','RightKnee_pn_pdf',...
                'leftLegs_pn_pdf','rightLegs_pn_pdf'});

%Assigning initial tokens to places used for holding tokes in each joints model
dyn.m0 = {...
          'pH_al' , 70, 'pH_be' , 63,'pH_ga' , 63,...
          'pT_al' , 30, 'pT_be' , 40,'pT_ga' , 90,...
          'pLE_al' , 130, 'pLE_be' , 170,'pLE_ga' , 170,...
          'pRE_al' , 130, 'pRE_be' , 170,'pRE_ga' , 170,...
          'pLW_ga' , 150,...
```

```
           'pRW_ga' , 150,...
           'pLK_al' , 50, 'pLK_be' , 45,'pLK_ga' , 90,...
           'pRK_al' , 50, 'pRK_be' , 45,'pRK_ga' , 90,....
           'pLA_ga' , 145,...
           'pRA_ga' , 145}; %Initial tokens


%Defining firing time for each transitions
dyn.ft = {...
             'tInstruct' , 1, ...
             ...%Head
            'tH_pos_al',1,'tHBack_pos_al',1,'tH_neg_al',1,'tHBack_neg_al',1,...
            'tH_pos_be',1,'tHBack_pos_be',1,'tH_neg_be',1,'tHBack_neg_be',1,...
            'tH_pos_ga',1,'tHBack_pos_ga',1,'tH_neg_ga',1,'tHBack_neg_ga',1 ,...
            ... %Trunk
            'tT_pos_al',1,'tTBack_pos_al',1,'tT_neg_al',1,'tTBack_neg_al',1,...
           'tT_pos_be',1,'tTBack_pos_be',1,'tT_neg_be',1,'tTBack_neg_be',1,...
            'tT_pos_ga',1,'tTBack_pos_ga',1,'tT_neg_ga',1,'tTBack_neg_ga',1 ,...
           ... %Left Elbow
            'tLE_pos_al',1,'tLEBack_pos_al',1,'tLE_neg_al',1,'tLEBack_neg_al',1,...
            'tLE_pos_be',1,'tLEBack_pos_be',1,'tLE_neg_be',1,'tLEBack_neg_be',1,...
            'tLE_pos_ga',1,'tLEBack_pos_ga',1,'tLE_neg_ga',1,'tLEBack_neg_ga',1,...
           ... %Right Elbow
            'tRE_pos_al',1,'tREBack_pos_al',1,'tRE_neg_al',1,'tREBack_neg_al',1,...
            'tRE_pos_be',1,'tREBack_pos_be',1,'tRE_neg_be',1,'tREBack_neg_be',1,...
            'tRE_pos_ga',1,'tREBack_pos_ga',1,'tRE_neg_ga',1,'tREBack_neg_ga',1,...
           ... %Left Wrist
            'tLW_pos_ga',1,'tLWBack_pos_ga',1,...
           ... %Right Wrist
            'tRW_pos_ga',1,'tRWBack_pos_ga',1,...
           ... %Left Knee
           'tLK_pos_al',1,'tLKBack_pos_al',1,'tLK_neg_al',1,'tLKBack_neg_al',1,...
           'tLK_pos_be',1,'tLKBack_pos_be',1,'tLK_neg_be',1,'tLKBack_neg_be',1,...
           'tLK_pos_ga',1,'tLKBack_pos_ga',1,'tLK_neg_ga',1,'tLKBack_neg_ga',1 ,...
           ... %Right Knee
           'tRK_pos_al',1,'tRKBack_pos_al',1,'tRK_neg_al',1,'tRKBack_neg_al',1,...
           'tRK_pos_be',1,'tRKBack_pos_be',1,'tRK_neg_be',1,'tRKBack_neg_be',1,...
           'tRK_pos_ga',1,'tRKBack_pos_ga',1,'tRK_neg_ga',1,'tRKBack_neg_ga',1,...
           ... %Left Ankle
           'tLA_neg_ga',1,'tLABack_neg_ga',1,...
           ... %Right Ankle
           'tRA_neg_ga',1,'tRABack_neg_ga',1 }; %Firing Time


%Passing initial information as packet to function 'initialdynamics'
pni = initialdynamics(pns,dyn);


%Simulation performed by function 'gpensim'
sim = gpensim(pni);

%prnss(sim);

%Display of flow of token in two figures where

%First set consist of Left Knee, Left Ankle, Right Elbow and Right Wrist
figure(1),plotpNew(sim,{ 'pLK_pos_al', 'pLK_neg_al','pLA_neg_ga', 'pRE_pos_al',
'pRE_neg_al', 'pRW_pos_ga'});
%Second set consist of Right Knee, Right Ankle, Left Elbow and Left Wrist
figure(2),plotpNew(sim,{ 'pRK_pos_al', 'pRK_neg_al','pRA_neg_ga', 'pLE_pos_al',
'pLE_neg_al', 'pLW_pos_ga'});
```

## 2. The instructor (pre-processor TDF) file

**tInstruct_pre.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The TDF (pre processor) for single robot mode
%This acts as a instructor or brain of robot
% Last modified: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [fire, transition] = tInstruct_pre(transition)

    global global_info;


    ctime = current_time();

        %Check each row of global_info.inputData which stores all values
        %From instruction file. Differentiate each task and perform work
        %Accoring to it
        if le(global_info.counter,global_info.rowSize(1)),
            while eq(ctime,
str2double(global_info.inputData{1}{global_info.counter})),
                      varTask = global_info.inputData{2}{global_info.counter};

                      %Task '3' is stochastic motion
                      if eq(varTask,'3'),
                          global_info.task = 3;
                          %To initialize all value
                           if ne( global_info.pre_task,3),
                                  initialize();
                            end
                          global_info.pre_task = 3; %assign current task as past
task

                          var1 = global_info.inputData{3}{global_info.counter};
                          var2 = global_info.inputData{4}{global_info.counter};
                          var3 = global_info.inputData{5}{global_info.counter};
                          var4 = global_info.inputData{6}{global_info.counter};
                          eval(['global_info.' var1 '_angle(' var2 ')=' var3]);
                          eval(['global_info.' var1 '_angle(4)=' var4]);

                       %Task '1' is walking motion
                       elseif eq(varTask,'1'),
                          global_info.task = 1;

                          %Initail values for Walking

                            %Right Leg
                            global_info.RK_angle = [20 0 0 0];
                            global_info.LE_angle = [-10 0 0 3];
                            %Left Leg
                            global_info.LK_angle = [-20 0 0 0];
                            global_info.RE_angle = [10 0 0 3];

                            global_info.pre_task = 1;

                       %Task '2' is running motion
                       elseif eq(varTask,'2'),
                            global_info.task = 2;
```

```
                              %Initial values for running

                              %Left Leg
                               global_info.RW_angle = [0 0 90 0];
                               global_info.RE_angle = [-30 0 0 7];
                               global_info.LK_angle = [25 0 0 7];
                               global_info.LA_angle = [0 0 -25 7];
                               %Right Leg
                               global_info.LW_angle = [0 0 120 0];
                               global_info.LE_angle = [5 0 0 40];
                               global_info.RK_angle = [-25 0 0 9];
                               global_info.RA_angle = [0 0 -35 6];

                               global_info.pre_task = 2;
                        end

                         %Increase counter and break if count is
                         %more than number of command lines
                         global_info.counter = global_info.counter + 1 ;
                         if gt(global_info.counter,global_info.rowSize(1)),
                              break;
                          end


            end
        end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Task '1' indicating Walking motion
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

     if eq(global_info.task,1),
            %For Walking

            %Right Leg

            %No of token in particular place
            b1 = get_place('pRK_pos_al');
            b2 = get_place('pRK_neg_al');

            if eq(b1.tokens,20),
                %Right Legs
                global_info.RK_angle = [-20 0 0 0];
                global_info.RA_angle = [0 0 0 6];
                %Left Hand
                global_info.LE_angle = [10 0 0 3];
                global_info.LW_angle = [0 0 0 3];

            elseif eq(b2.tokens,20),
                %Right Legs
                global_info.RK_angle = [20 0 0 0];
                 global_info.RA_angle = [0 0 0 1];
                 %Left Hand
                global_info.LE_angle = [-10 0 0 3];
                global_info.LW_angle = [0 0 0 1];

            elseif and(eq(b2.tokens,5),lt(global_info.RK_angle(1),0)),
                %Right Legs
                global_info.RA_angle = [0 0 -15 1];
            elseif and(eq(b1.tokens,1),gt(global_info.RK_angle(1),0)),
                %Right Legs
                global_info.RA_angle = [0 0 -5 6];
                %Left Hand
                 global_info.LW_angle = [0 0 10 3];
             elseif and(eq(b2.tokens,1),lt(global_info.RK_angle(1),0)),
                 %Left Hand
                global_info.LW_angle = [0 0 20 1];
            end

            %Left Leg

            %No of token in particular place
            b3 = get_place('pLK_pos_al');
            b4 = get_place('pLK_neg_al');

            if eq(b3.tokens,20),
                %Left Legs
                global_info.LK_angle = [-20 0 0 0];
                global_info.LA_angle = [0 0 0 6];
                %Right Hand
                global_info.RE_angle = [10 0 0 3];
                global_info.RW_angle = [0 0 0 3];

             elseif eq(b4.tokens,20),
                %Left Legs
                global_info.LK_angle = [20 0 0 0];
                 global_info.LA_angle = [0 0 0 1];
                 %Right Hand
                global_info.RE_angle = [-10 0 0 3];
                global_info.RW_angle = [0 0 0 1];

            elseif and(eq(b4.tokens,5),lt(global_info.LK_angle(1),0)),
                %Left Legs
                global_info.LA_angle = [0 0 -15 1];
            elseif and(eq(b3.tokens,1),gt(global_info.LK_angle(1),0)),
                %Left Legs
                global_info.LA_angle = [0 0 -5 6];
                %Right Hand
                global_info.RW_angle = [0 0 10 3];
            elseif and(eq(b4.tokens,1),lt(global_info.LK_angle(1),0)),
                %Right Hand
                global_info.RW_angle = [0 0 20 1];
            end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Task '2' indicating Running motion
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    elseif eq(global_info.task,2),

        %For running

        %Left Legs

        %No of token in particular place
            b1 = get_place('pLK_pos_al'); %Gets total tokens as place LK pos
            b2 = get_place('pLK_neg_al'); %Gets total tokens as place LK neg


            if eq(b1.tokens,25),
                %$Right Hand
                global_info.RE_angle = [5 0 0 2];
                global_info.RW_angle = [0 0 120 2];
                global_info.LK_angle = [-25 0 0 1];
                global_info.LA_angle = [0 0 -35 8];
            elseif eq(b2.tokens,25),
                global_info.RE_angle = [-30 0 0 2];
                global_info.RW_angle = [0 0 90 2];
                global_info.LK_angle = [25 0 0 1];
                global_info.LA_angle = [0 0 -25 8];
            end

           %Right Legs

          %No of token in particular place
            b3 = get_place('pRK_pos_al');
            b4 = get_place('pRK_neg_al');


            if eq(b3.tokens,25),
                %$Right Hand
                global_info.LE_angle = [5 0 0 2];
                global_info.LW_angle = [0 0 120 2];
                global_info.RK_angle = [-25 0 0 1];
                global_info.RA_angle = [0 0 -35 8];
            elseif eq(b4.tokens,25),
                global_info.LE_angle = [-30 0 0 2];
                global_info.LW_angle = [0 0 90 2];
                global_info.RK_angle = [25 0 0 1];
                global_info.RA_angle = [0 0 -25 8];
            end


    end

       fire = 1 ;


   end
```

## 3. The initialization file

**Initialize.m**

```
%Function called to initialize all angles

function []= initialize()
    global global_info;

    %initializing all the joints
    %First element is alpha angle
    %Second is beta angle
    %Third is gamma angle
    %Final element is firing time
    global_info.H_angle = [0 0 0 0]; %Head
    global_info.T_angle = [0 0 0 0]; %Trunk
    global_info.LE_angle = [0 0 0 0];%Left Elbow
    global_info.RE_angle = [0 0 0 0];%Right Elbow
    global_info.LW_angle = [0 0 0 0];%Left Wrist
    global_info.RW_angle = [0 0 0 0];%Right Wrist
    global_info.LK_angle = [0 0 0 0];%Left Knee
    global_info.RK_angle = [0 0 0 0];%Right Knee
    global_info.LA_angle = [0 0 0 0];%Left Ankle
    global_info.RA_angle = [0 0 0 0];%Right Ankle

    %First variable is counter for angle moves in postive direction,
    %second variable is to record firing time rather than sampling time and
    %Third variable is counter for angle moves in opposite direction,
    global_info.var_RA = [0 0 0];  %For Right hand(ankle)
    global_info.var_LA = [0 0 0]; %For Left leg (ankle)
    global_info.var_RW = [0 0 0]; %For Right hand(wrist)
    global_info.var_LW = [0 0 0]; %For Left hand(wrist)
    global_info.var_LEp = [0 0 0]; %For Left leg (ankle) - positive direction
    global_info.var_LEn = [0 0 0]; %For Left leg (ankle) - negative direction
    global_info.var_REp = [0 0 0]; %For Right hand(wrist) - positive direction
    global_info.var_REn = [0 0 0]; %For Right hand(wrist) - negative direction
    global_info.var_LKp = [0 0 0]; %For Right hand(wrist) - positive direction
    global_info.var_LKn = [0 0 0]; %For Right hand(wrist) - negative direction
    global_info.var_RKp = [0 0 0]; %For Right hand(wrist) - positive direction
    global_info.var_RKn = [0 0 0]; %For Left hand(wrist) - negative direction

end
```

## 4. The PDF (Petri Net Definition File)

**RightKnee_pn_pdf.m**

Since the PDFs have same way of implementation with only different variable names. Only one PDF of joint RK (Right knee) is displayed in this section. There are total of 11 PDF files in this implementation of single robot. One of the 11 PDF defines instructor transition which has one transition, no places and no arc so always enabled.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The PDF for single robot mode representing Right Knee joint
%with all three angle
% Last modified: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [pns] = RightKnee_pn_pdf()

    pns.PN_name= 'The Right Knee of the robot';
    pns.set_of_Ps = {'pRK_al','pRK_pos_al','pRK_neg_al',...
                     'pRK_be','pRK_pos_be','pRK_neg_be',...
                     'pRK_ga','pRK_pos_ga','pRK_neg_ga'};

    pns.set_of_Ts =
{'tRK_pos_al','tRKBack_pos_al','tRK_neg_al','tRKBack_neg_al',...

'tRK_pos_be','tRKBack_pos_be','tRK_neg_be','tRKBack_neg_be',...
                     'tRK_pos_ga','tRKBack_pos_ga','tRK_neg_ga','tRKBack_neg_ga'};

    pns.set_of_As = {'pRK_al', 'tRK_pos_al',1 ,'pRK_al', 'tRK_neg_al',1 ,...
                    'tRK_pos_al', 'pRK_pos_al',1 ,'tRK_neg_al', 'pRK_neg_al',1 ,...
                    'pRK_pos_al', 'tRKBack_pos_al',1 ,'tRKBack_pos_al', 'pRK_al',1
,...
                    'pRK_neg_al', 'tRKBack_neg_al',1 ,'tRKBack_neg_al',
'pRK_al',1,...
                    ...
                    'pRK_be', 'tRK_pos_be',1 ,'pRK_be', 'tRK_neg_be',1 ,...
                    'tRK_pos_be', 'pRK_pos_be',1 ,'tRK_neg_be', 'pRK_neg_be',1 ,...
                    'pRK_pos_be', 'tRKBack_pos_be',1 ,'tRKBack_pos_be', 'pRK_be',1
,...
                    'pRK_neg_be', 'tRKBack_neg_be',1 ,'tRKBack_neg_be', 'pRK_be',1
,...
                    ...
                    'pRK_ga', 'tRK_pos_ga',1 ,'pRK_ga', 'tRK_neg_ga',1 ,...
```

```
                        'tRK_pos_ga', 'pRK_pos_ga',1 ,'tRK_neg_ga', 'pRK_neg_ga',1 ,...
                        'pRK_pos_ga', 'tRKBack_pos_ga',1 ,'tRKBack_pos_ga', 'pRK_ga',1
,...
                        'pRK_neg_ga', 'tRKBack_neg_ga',1 ,'tRKBack_neg_ga', 'pRK_ga',1
};

    pns.set_of_Is ={'pRK_pos_al', 'tRK_pos_al',50 ,'pRK_pos_al', 'tRK_neg_al',1
,...
                        'pRK_neg_al', 'tRK_neg_al',45 ,'pRK_neg_al', 'tRK_pos_al',1,...
...
                        'pRK_pos_be', 'tRK_pos_be',30 ,'pRK_pos_be', 'tRK_neg_be',1
,...
                        'pRK_neg_be', 'tRK_neg_be',45 ,'pRK_neg_be', 'tRK_pos_be',1,...
...
                        'pRK_pos_ga', 'tRK_pos_ga',90 ,'pRK_pos_ga', 'tRK_neg_ga',1
,...
                        'pRK_neg_ga', 'tRK_neg_ga',15 ,'pRK_neg_ga', 'tRK_pos_ga',1};
```

## 5. TDF for pre processor of transitions

Each model represents an angle of a joint. These models have four transitions. As described in implementation each angle of all joints work similar to each other. So only TDFs for joint RK (Right knee) in **α** direction is included in this section. The suffix shown as '*al*' refers to **α** direction. Similarly suffix '*be*' refers to **β** and '*ga*' refers to **γ** in this implementation. In total, 40 TDFs are used to represent all joints and their angles as full package.

**<u>tRK_pos_al_pre.m</u>**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The TDF (pre-processor) of Joint -Right knee
%in alpha angle where movement is towards postive direction
% Last modified: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [fire, transition] = tRK_pos_al_pre(transition)

    global global_info;
    b1 = get_place('pRK_pos_al');
    ctime = current_time();

    %Check tokens in postive token place and designed input degree value
    %to fire
    if and(gt(global_info.RK_angle(1),0), lt(b1.tokens,global_info.RK_angle(1))),
            fire = 1;
    else
            fire = 0;
            global_info.var_RKp(1) = 0;
    end

    %For variable timing to fire with respect to firing time
    %rather than sampling time
     if and(ne(global_info.var_RKp(2),ctime),eq(fire,1)), %if uniquie time
                   global_info.var_RKp(1) = global_info.var_RKp(1) +1;
```

```
                    global_info.var_RKp(2) = ctime;
      end

      %Assigng value -1 if presviously assigned 0 to avoid mod issue below
      if eq(global_info.RK_angle(4),0),
          global_info.RK_angle(4) = -1;
      end;

       %Firing when module output is zero
      %Meaning fire token per every input variable input firing time(RK_angle(4))
      %var_RKn(1) is counter which stores each firing
      modul = mod(global_info.var_RKp(1),global_info.RK_angle(4));
       if and(le(modul,0),eq(fire,1)), %For moudule less tha equal 0 and if fire is
true
             fire = 1;
             global_info.var_RKp(1) = 0;
       else
             fire = 0 ;
       end

end
```

## tRK_neg_al_pre.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The TDF (pre-processor) of Joint -Right knee
%in alpha angle where movement is towards negative direction
% Last modified: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [fire, transition] = tRK_neg_al_pre(transition)

    global global_info;
    b1 = get_place('pRK_neg_al');
    ctime = current_time();

    %Check tokens in negative token place and designed input degree value
    %to fire
    if and(lt(global_info.RK_angle(1),0), gt((b1.tokens * -
1),global_info.RK_angle(1))),
            fire = 1;
    else
            fire = 0;
            global_info.var_RKn(1) = 0;
    end

    %For variable timing to fire with respect to firing time
    %rather than sampling time
    if and(ne(global_info.var_RKn(2),ctime),eq(fire,1)), %if uniquie time
                    global_info.var_RKn(1) = global_info.var_RKn(1) +1;
                    global_info.var_RKn(2) = ctime;
     end

     %Assigng value -1 if presviously assigned 0 to avoid mod issue below
     if eq(global_info.RK_angle(4),0),
         global_info.RK_angle(4) = -1;
     end;

     %Firing when module output is zero
     %Meaning fire token per every input variable input firing time(RK_angle(4))
     %var_RKn(1) is counter which stores each firing
     modul = mod(global_info.var_RKn(1),global_info.RK_angle(4));
      if and(le(modul,0),eq(fire,1)), %For moudule less tha equal 0 and if fire is
true
             fire = 1;
             global_info.var_RKn(1) = 0;
      else
             fire = 0 ;
      end

end
```

## tRKBack_pos_al_pre.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The TDF (pre-processor) of Joint -Right knee
%in alpha angle where tokens are send back to storing place
%from postive degree place holder
% Last modifided: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [fire, transition] = tRKBack_pos_al_pre(transition)

    global global_info;
    b1 = get_place('pRK_pos_al');
    ctime = current_time();

    %Check tokens in postitive token place and designed input degree value
    %to fire if need change then fires back
    if and(gt(b1.tokens,global_info.RK_angle(1)),ge(global_info.RK_angle(1),0)),
            fire = 1 ;
    elseif le(global_info.RK_angle(1),0),
            fire = 1;
    else
            fire=0;
            global_info.var_RKp(3) = 0;
    end

    %For variable timing to fire with respect to firing time
    %rather than sampling time
    if and(ne(global_info.var_RKp(2),ctime),eq(fire,1)), %if uniquie time
                  global_info.var_RKp(3) = global_info.var_RKp(3) +1;
                  global_info.var_RKp(2) = ctime;
    end

    %Assigng value -1 if presviously assigned 0 to avoid mod issue below
    if eq(global_info.RK_angle(4),0),
         global_info.RK_angle(4) = -1;
     end;

     %Firing when module output is zero
     %Meaning fire token per every input variable input firing time(RK_angle(4))
     %var_RKn(1) is counter which stores each firing
    modul = mod(global_info.var_RKp(3),global_info.RK_angle(4));
      if and(le(modul,0),eq(fire,1)), %For moudule less tha equal 0, if enabled
             fire = 1;
             global_info.var_RKp(3) = 0;
      else
             fire = 0 ;
      end

end
```

## tRKBack_neg_al_pre.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The TDF (pre-processor) of Joint -Right knee
%in alpha angle where tokens are send back to storing place
%from negative degree place holder
% Last modifided: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [fire, transition] = tRKBack_neg_al_pre(transition)

    global global_info;
    b1 = get_place('pRK_neg_al');
    ctime = current_time();

    %Check tokens in negative token place and designed input degree value
    %to fire if need change then fires back
    if and(lt((b1.tokens * -
1),global_info.RK_angle(1)),le(global_info.RK_angle(1),0)),
            fire = 1 ;
    elseif ge(global_info.RK_angle(1),0),
            fire = 1;
    else
            fire=0;
            global_info.var_RKn(3) = 0;
    end

    %For variable timing to fire with respect to firing time
    %rather than sampling time
    if and(ne(global_info.var_RKn(2),ctime),eq(fire,1)), %if uniquie time
                  global_info.var_RKn(3) = global_info.var_RKn(3) +1;
                  global_info.var_RKn(2) = ctime;
    end

     %Assigng value -1 if presviously assigned 0 to avoid mod issue below
    if eq(global_info.RK_angle(4),0),
         global_info.RK_angle(4) = -1;
     end;

     %Firing when module output is zero
     %Meaning fire token per every input variable input firing time(RK_angle(4))
     %var_RKn(1) is counter which stores each firing
    modul = mod(global_info.var_RKn(3),global_info.RK_angle(4));
      if and(le(modul,0),eq(fire,1)), %For moudule less tha equal 0, if enabled
            fire = 1;
            global_info.var_RKn(3) = 0;
      else
            fire = 0 ;
      end

end
```

## C2. Source code of Multi robot mode

### 1. The main simulation file (MSF)

**robot_swarm.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The MSF for mutiple robot mode
%This implementation consist of two robots
%Robot1 and robot2
% Last modified: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all; clc;

global global_info;
global_info.DELTA_TIME= 0.5; %timer increment value is 0.5 sec(high sampling rate)
global_info.STOP_AT = 300; %stop sim after 300 time


%Identify cyrrent folder
%here 'pwd' gives current path set
%'mfilename' give current .m file location
currentFolder = mfilename('fullpath'); %disp extra robot , not sure abt it
addPath1 = strcat(currentFolder,'1') ;
addPath2 = strcat(currentFolder,'2') ;
addpath(addPath1,addPath2);


 %To initialize all angles to zero
  r1_initialize(); %For robot 1
  r2_initialize(); %For robot 2


%counter used by instructor, begins from second line to avoid the title
%first line in instruction file
global_info.counter = 2;

%For Inputing data from a file : 'instructFile.txt'
fileID = fopen('instructFile.txt','r');
global_info.inputData = textscan(fileID,'%s %s %s %s %s %s %s');
fclose(fileID);

 %obtain row size of an array
 global_info.rowSize=cellfun(@length,global_info.inputData);

%Variable indicating diffrent task in cell arry where
%First element is for robot1 and second for robot 2
global_info.task = [3,3];




%Passing of PDFs to function 'pnstruct'
pns = pnstruct({...
                'instructor_pn_pdf',...
...             %Robot1
                'r1_instructor_pn_pdf',...
...%                 'r1_head_pn_pdf','r1_trunk_pn_pdf',
                'r1_LeftElbow_pn_pdf','r1_RightElbow_pn_pdf',...
               'r1_leftWrist_pn_pdf','r1_rightWrist_pn_pdf',...
                'r1_LeftKnee_pn_pdf','r1_RightKnee_pn_pdf',...
```

```
                 'r1_leftLegs_pn_pdf','r1_rightLegs_pn_pdf',...
...                %Robot2
                'r2_instructor_pn_pdf',...
...%                  'r2_head_pn_pdf','r2_trunk_pn_pdf',
                 'r2_LeftElbow_pn_pdf','r2_RightElbow_pn_pdf',...
                'r2_leftWrist_pn_pdf','r2_rightWrist_pn_pdf',...
                 'r2_LeftKnee_pn_pdf','r2_RightKnee_pn_pdf',...
                 'r2_leftLegs_pn_pdf','r2_rightLegs_pn_pdf'});

%Assiginig initial tokens to places used for holding tokes in each joints model
dyn.m0 = {...
...          %Robot1
          'pr1_H_al' , 70, 'pr1_H_be' , 63,'pr1_H_ga' , 63,...
          'pr1_T_al' , 30, 'pr1_T_be' , 40,'pr1_T_ga' , 90,...
           'pr1_LE_al' , 130, 'pr1_LE_be' , 170,'pr1_LE_ga' , 170,...
           'pr1_RE_al' , 130, 'pr1_RE_be' , 170,'pr1_RE_ga' , 170,...
           'pr1_LW_ga' , 150,...
           'pr1_RW_ga' , 150,...
          'pr1_LK_al' , 50, 'pr1_LK_be' , 45,'pr1_LK_ga' , 90,...
          'pr1_RK_al' , 50, 'pr1_RK_be' , 45,'pr1_RK_ga' , 90,....
          'pr1_LA_ga' , 145,...
          'pr1_RA_ga' , 145,...
...          %Robot2
           'pr2_H_al' , 70, 'pr2_H_be' , 63,'pr2_H_ga' , 63,...
          'pr2_T_al' , 30, 'pr2_T_be' , 40,'pr2_T_ga' , 90,...
           'pr2_LE_al' , 130, 'pr2_LE_be' , 170,'pr2_LE_ga' , 170,...
           'pr2_RE_al' , 130, 'pr2_RE_be' , 170,'pr2_RE_ga' , 170,...
           'pr2_LW_ga' , 150,...
           'pr2_RW_ga' , 150,...
          'pr2_LK_al' , 50, 'pr2_LK_be' , 45,'pr2_LK_ga' , 90,...
          'pr2_RK_al' , 50, 'pr2_RK_be' , 45,'pr2_RK_ga' , 90,....
          'pr2_LA_ga' , 145,...
          'pr2_RA_ga' , 145}; %Initial tokens



%Defining firing time for each transitions
dyn.ft = {...
            'tInstruct' , 1, ...
            ...%Robot1
            'tr1_Instruct',1,...
            ...%Head

'tr1_H_pos_al',1,'tr1_HBack_pos_al',1,'tr1_H_neg_al',1,'tr1_HBack_neg_al',1,...

'tr1_H_pos_be',1,'tr1_HBack_pos_be',1,'tr1_H_neg_be',1,'tr1_HBack_neg_be',1,...

'tr1_H_pos_ga',1,'tr1_HBack_pos_ga',1,'tr1_H_neg_ga',1,'tr1_HBack_neg_ga',1 ,...
           ... %Trunk

'tr1_T_pos_al',1,'tr1_TBack_pos_al',1,'tr1_T_neg_al',1,'tr1_TBack_neg_al',1,...

'tr1_T_pos_be',1,'tr1_TBack_pos_be',1,'tr1_T_neg_be',1,'tr1_TBack_neg_be',1,...

'tr1_T_pos_ga',1,'tr1_TBack_pos_ga',1,'tr1_T_neg_ga',1,'tr1_TBack_neg_ga',1 ,...
          ... %Left Elbow

'tr1_LE_pos_al',1,'tr1_LEBack_pos_al',1,'tr1_LE_neg_al',1,'tr1_LEBack_neg_al',1,...

'tr1_LE_pos_be',1,'tr1_LEBack_pos_be',1,'tr1_LE_neg_be',1,'tr1_LEBack_neg_be',1,...

'tr1_LE_pos_ga',1,'tr1_LEBack_pos_ga',1,'tr1_LE_neg_ga',1,'tr1_LEBack_neg_ga',1,...
          ... %Right Elbow

'tr1_RE_pos_al',1,'tr1_REBack_pos_al',1,'tr1_RE_neg_al',1,'tr1_REBack_neg_al',1,...
```

```
'tr1_RE_pos_be',1,'tr1_REBack_pos_be',1,'tr1_RE_neg_be',1,'tr1_REBack_neg_be',1,...

'tr1_RE_pos_ga',1,'tr1_REBack_pos_ga',1,'tr1_RE_neg_ga',1,'tr1_REBack_neg_ga',1,...
          ... %Left Wrist
           'tr1_LW_pos_ga',1,'tr1_LWBack_pos_ga',1,...
          ... %Right Wrist
           'tr1_RW_pos_ga',1,'tr1_RWBack_pos_ga',1,...
          ... %Left Knee

'tr1_LK_pos_al',1,'tr1_LKBack_pos_al',1,'tr1_LK_neg_al',1,'tr1_LKBack_neg_al',1,...

'tr1_LK_pos_be',1,'tr1_LKBack_pos_be',1,'tr1_LK_neg_be',1,'tr1_LKBack_neg_be',1,...

'tr1_LK_pos_ga',1,'tr1_LKBack_pos_ga',1,'tr1_LK_neg_ga',1,'tr1_LKBack_neg_ga',1
,...
          ... %Right Knee

'tr1_RK_pos_al',1,'tr1_RKBack_pos_al',1,'tr1_RK_neg_al',1,'tr1_RKBack_neg_al',1,...

'tr1_RK_pos_be',1,'tr1_RKBack_pos_be',1,'tr1_RK_neg_be',1,'tr1_RKBack_neg_be',1,...

'tr1_RK_pos_ga',1,'tr1_RKBack_pos_ga',1,'tr1_RK_neg_ga',1,'tr1_RKBack_neg_ga',1,...
          ... %Left Ankle
          'tr1_LA_neg_ga',1,'tr1_LABack_neg_ga',1,...
          ... %Right Ankle
          'tr1_RA_neg_ga',1,'tr1_RABack_neg_ga',1,...
       ...%Robot2
       'tr2_Instruct',1,...
            ...%Head

'tr2_H_pos_al',1,'tr2_HBack_pos_al',1,'tr2_H_neg_al',1,'tr2_HBack_neg_al',1,...

'tr2_H_pos_be',1,'tr2_HBack_pos_be',1,'tr2_H_neg_be',1,'tr2_HBack_neg_be',1,...

'tr2_H_pos_ga',1,'tr2_HBack_pos_ga',1,'tr2_H_neg_ga',1,'tr2_HBack_neg_ga',1 ,...
           ... %Trunk

'tr2_T_pos_al',1,'tr2_TBack_pos_al',1,'tr2_T_neg_al',1,'tr2_TBack_neg_al',1,...

'tr2_T_pos_be',1,'tr2_TBack_pos_be',1,'tr2_T_neg_be',1,'tr2_TBack_neg_be',1,...

'tr2_T_pos_ga',1,'tr2_TBack_pos_ga',1,'tr2_T_neg_ga',1,'tr2_TBack_neg_ga',1 ,...
          ... %Left Elbow

'tr2_LE_pos_al',1,'tr2_LEBack_pos_al',1,'tr2_LE_neg_al',1,'tr2_LEBack_neg_al',1,...

'tr2_LE_pos_be',1,'tr2_LEBack_pos_be',1,'tr2_LE_neg_be',1,'tr2_LEBack_neg_be',1,...

'tr2_LE_pos_ga',1,'tr2_LEBack_pos_ga',1,'tr2_LE_neg_ga',1,'tr2_LEBack_neg_ga',1,...
          ... %Right Elbow

'tr2_RE_pos_al',1,'tr2_REBack_pos_al',1,'tr2_RE_neg_al',1,'tr2_REBack_neg_al',1,...

'tr2_RE_pos_be',1,'tr2_REBack_pos_be',1,'tr2_RE_neg_be',1,'tr2_REBack_neg_be',1,...

'tr2_RE_pos_ga',1,'tr2_REBack_pos_ga',1,'tr2_RE_neg_ga',1,'tr2_REBack_neg_ga',1,...
          ... %Left Wrist
           'tr2_LW_pos_ga',1,'tr2_LWBack_pos_ga',1,...
          ... %Right Wrist
           'tr2_RW_pos_ga',1,'tr2_RWBack_pos_ga',1,...
          ... %Left Knee

'tr2_LK_pos_al',1,'tr2_LKBack_pos_al',1,'tr2_LK_neg_al',1,'tr2_LKBack_neg_al',1,...

'tr2_LK_pos_be',1,'tr2_LKBack_pos_be',1,'tr2_LK_neg_be',1,'tr2_LKBack_neg_be',1,...
```

```
'tr2_LK_pos_ga',1,'tr2_LKBack_pos_ga',1,'tr2_LK_neg_ga',1,'tr2_LKBack_neg_ga',1
,...
          ... %Right Knee

'tr2_RK_pos_al',1,'tr2_RKBack_pos_al',1,'tr2_RK_neg_al',1,'tr2_RKBack_neg_al',1,...

'tr2_RK_pos_be',1,'tr2_RKBack_pos_be',1,'tr2_RK_neg_be',1,'tr2_RKBack_neg_be',1,...

'tr2_RK_pos_ga',1,'tr2_RKBack_pos_ga',1,'tr2_RK_neg_ga',1,'tr2_RKBack_neg_ga',1,...
          ... %Left Ankle
          'tr2_LA_neg_ga',1,'tr2_LABack_neg_ga',1,...
          ... %Right Ankle
          'tr2_RA_neg_ga',1,'tr2_RABack_neg_ga',1 }; %Firing Time


%Passing initial information as packet to function 'initialdynamics'
pni = initialdynamics(pns,dyn);

%Simulation performed by function 'gpensim'
sim = gpensim(pni);

%prnss(sim);


%Display of flow of token in two figures each where

 %For Robot 1
     %First set consist of Left Knee, Left Ankle, Right Elbow and Right Wrist
     figure(1),plotpNew(sim,{ 'pr1_LK_pos_al', 'pr1_LK_neg_al','pr1_LA_neg_ga',
'pr1_RE_pos_al', 'pr1_RE_neg_al', 'pr1_RW_pos_ga'});
     %Second set consist of Right Knee, Right Ankle, Left Elbow and Left Wrist
     figure(2),plotpNew(sim,{ 'pr1_RK_pos_al', 'pr1_RK_neg_al','pr1_RA_neg_ga',
'pr1_LE_pos_al', 'pr1_LE_neg_al', 'pr1_LW_pos_ga'});

 %For Robot 2
     %First set consist of Left Knee, Left Ankle, Right Elbow and Right Wrist
     figure(3),plotpNew(sim,{ 'pr2_LK_pos_al', 'pr2_LK_neg_al','pr2_LA_neg_ga',
'pr2_RE_pos_al', 'pr2_RE_neg_al', 'pr2_RW_pos_ga'});
     %Second set consist of Right Knee, Right Ankle, Left Elbow and Left Wrist
     figure(4),plotpNew(sim,{ 'pr2_RK_pos_al', 'pr2_RK_neg_al','pr2_RA_neg_ga',
'pr2_LE_pos_al', 'pr2_LE_neg_al', 'pr2_LW_pos_ga'});
```

## 2. The instructor file

**tInstruct_pre.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The TDF (pre processor) for multiple robot mode
%This acts as a main instructor for both robot1 and robot 2
% Last modifided: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [fire, transition] = tInstruct_pre(transition)

    global global_info;
    ctime = current_time();

        %Check each row of global_info.inputData which stores all values
        %From instruction file. Differentiate each task and perform work
        %Accoring to it
        if le(global_info.counter,global_info.rowSize(1)),
            while eq(ctime,
str2double(global_info.inputData{1}{global_info.counter})),
                      varTask = global_info.inputData{3}{global_info.counter};
                      varR = global_info.inputData{2}{global_info.counter};

                      %Task '3' is stochastic motion
                      if eq(varTask,'3'),

                          %Initialize for different robot when the previous
                          %task was to run or walk
                          if
and(eq(varR,'1'),or(eq(global_info.task(1),1),eq(global_info.task(1),2))),
                              r1_initialize();
                          elseif
and(eq(varR,'2'),or(eq(global_info.task(2),1),eq(global_info.task(2),2))),
                              r2_initialize();
                          end

                          var1 = global_info.inputData{4}{global_info.counter};
                          var2 = global_info.inputData{5}{global_info.counter};
                          var3 = global_info.inputData{6}{global_info.counter};
                          var4 = global_info.inputData{7}{global_info.counter};
                          eval(['global_info.r' varR '_' var1 '_angle(' var2 ')='
var3]);
                          eval(['global_info.r' varR '_' var1 '_angle(4)=' var4]);


global_info.task(str2double(global_info.inputData{2}{global_info.counter})) = 3;

                       %Task '1' is walking motion
                       elseif eq(varTask,'1'),
                          %Initial values for walking

                            %Right Leg
                            eval(['global_info.r' varR '_RK_angle =[20 0 0 0];']);
                            eval(['global_info.r' varR '_LE_angle=[-10 0 0 3];']);
                            %Left Leg
                            eval(['global_info.r' varR '_LK_angle =[-20 0 0 0];']);
                            eval(['global_info.r' varR '_RE_angle=[10 0 0 3];']);


global_info.task(str2double(global_info.inputData{2}{global_info.counter})) = 1;

                       %Task '2' is running motion
                       elseif eq(varTask,'2'),
                            %Initial values for running

                              %Left Leg
                                eval(['global_info.r' varR '_RW_angle =[0 0 90
0];']);

                                eval(['global_info.r' varR '_RE_angle=[-30 0 0
7];']);

                                eval(['global_info.r' varR '_LK_angle =[25 0 0
7];']);

                                eval(['global_info.r' varR '_LA_angle=[0 0 -25
7];']);
                              %Right Leg
                                eval(['global_info.r' varR '_LW_angle =[0 0 120
0];']);
```

```
                                eval(['global_info.r' varR '_LE_angle=[5 0 0 
40];']);
                                eval(['global_info.r' varR '_RK_angle =[-25 0 0 
9];']);
                                eval(['global_info.r' varR '_RA_angle=[0 0 -35 
6];']);


global_info.task(str2double(global_info.inputData{2}{global_info.counter})) = 2;
                      end

                       %Increase counter and break if count is
                       %more than number of command lines
                       global_info.counter = global_info.counter + 1 ;
                       if gt(global_info.counter,global_info.rowSize(1)), %Break if 
count if more than number of commands
                            break;
                        end


            end
        end

       fire = 1 ;

end
```

## 3. Initialize file

There are two files for initialization ‘*r1_initialize.m*’ for robot 1 and ‘*r2_initialize.m*’ for robot 2. They are similar to each other. Only difference is variable names. So, only initialization for ‘*robot 1*’ is shown below.

### r1_initialize.m

```
%Function called to initialize all angles of robot 1

function []= r1_initialize()
    global global_info;

    %initializing all the joints
    %First element is alpha angle
    %Second is beta angle
    %Third is gamma angle
    %Final element is firing time
    global_info.r1_H_angle = [0 0 0];   %Head
    global_info.r1_T_angle = [0 0 0];   %Trunk
    global_info.r1_LE_angle = [0 0 0 0];%Left Elbow
    global_info.r1_RE_angle = [0 0 0 0];%Right Elbow
    global_info.r1_LW_angle = [0 0 0 0];%Left Wrist
    global_info.r1_RW_angle = [0 0 0 0];%Right Wrist
    global_info.r1_LK_angle = [0 0 0 0];%Left Knee
    global_info.r1_RK_angle = [0 0 0 0];%Right Knee
    global_info.r1_LA_angle = [0 0 0 0];%Left Ankle
    global_info.r1_RA_angle = [0 0 0 0];%Right Ankle

   %First variable is counter for angle moves in postive direction,
    %second variable is to record firing time rather than sampling time and
    %Third variable is counter for angle moves in opposite direction,
    global_info.r1_var_RA = [0 0 0];  %For Right hand(ankle)
    global_info.r1_var_LA = [0 0 0]; %For Left leg (ankle)
    global_info.r1_var_RW = [0 0 0]; %For Right hand(wrist)
    global_info.r1_var_LW = [0 0 0]; %For LEft hand(wrist)
    global_info.r1_var_LEp = [0 0 0]; %For LEft leg (ankle)- positive direction
    global_info.r1_var_LEn = [0 0 0]; %For LEft leg (ankle)- negative direction
    global_info.r1_var_REp = [0 0 0]; %For Right hand(wrist)- positive direction
    global_info.r1_var_REn = [0 0 0]; %For Right hand(wrist)- negative direction
    global_info.r1_var_LKp = [0 0 0]; %For Right hand(wrist)- positive direction
    global_info.r1_var_LKn = [0 0 0]; %For Right hand(wrist)- negative direction
    global_info.r1_var_RKp = [0 0 0]; %For Right hand(wrist)- positive direction
    global_info.r1_var_RKn = [0 0 0]; %For LEft hand(wrist)- negative direction

end
```

## 4. Individual instruct file

Both robots utilizes separate instruct file under main instruct file. Due to their similarities; only instruct file of ‘*robot 1*’ is shown here.

**tr1_Instruct_pre.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The TDF (pre processor) for multi robot mode
%This acts as a instructor or brain of robot 1
% Last modified: 19/06/2015
%by Bikee Maharjan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [fire, transition] = tr1_Instruct_pre(transition)
    global global_info;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Taask '1' indicating Walking motion
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if eq(global_info.task(1),1),

            %For Walking

            %Right Leg

            %No of token in particular place
            b1 = get_place('pr1_RK_pos_al');
            b2 = get_place('pr1_RK_neg_al');


            if eq(b1.tokens,20),
                %Right Legs
                global_info.r1_RK_angle = [-20 0 0 0];
                global_info.r1_RA_angle = [0 0 0 6];
                %Left Hand
                global_info.r1_LE_angle = [10 0 0 3];
                global_info.r1_LW_angle = [0 0 0 3];

            elseif eq(b2.tokens,20),
                %Right Legs
                global_info.r1_RK_angle = [20 0 0 0];
                 global_info.r1_RA_angle = [0 0 0 1];
                 %Left Hand
                global_info.r1_LE_angle = [-10 0 0 3];
                global_info.r1_LW_angle = [0 0 0 1];

            elseif and(eq(b2.tokens,5),lt(global_info.r1_RK_angle(1),0)),
                %Right Legs
                global_info.r1_RA_angle = [0 0 -15 1];
            elseif and(eq(b1.tokens,1),gt(global_info.r1_RK_angle(1),0)),
                %Right Legs
                global_info.r1_RA_angle = [0 0 -5 6];
                %Left Hand
                 global_info.r1_LW_angle = [0 0 10 3];
             elseif and(eq(b2.tokens,1),lt(global_info.r1_RK_angle(1),0)),
                 %Left Hand
                global_info.r1_LW_angle = [0 0 20 1];
            end



            %Left Leg

            %No of token in particular place
            b3 = get_place('pr1_LK_pos_al');
            b4 = get_place('pr1_LK_neg_al');

            if eq(b3.tokens,20),
                %Left Legs
                global_info.r1_LK_angle = [-20 0 0 0];
                global_info.r1_LA_angle = [0 0 0 6];
                %Right Hand
                global_info.r1_RE_angle = [10 0 0 3];
                global_info.r1_RW_angle = [0 0 0 3];

             elseif eq(b4.tokens,20),
                %Left Legs
                global_info.r1_LK_angle = [20 0 0 0];
                 global_info.r1_LA_angle = [0 0 0 1];
                 %Right Hand
                global_info.r1_RE_angle = [-10 0 0 3];
                global_info.r1_RW_angle = [0 0 0 1];

            elseif and(eq(b4.tokens,5),lt(global_info.r1_LK_angle(1),0)),
                %Left Legs
                global_info.r1_LA_angle = [0 0 -15 1];
            elseif and(eq(b3.tokens,1),gt(global_info.r1_LK_angle(1),0)),
                %Left Legs
                global_info.r1_LA_angle = [0 0 -5 6];
                %Right Hand
                global_info.r1_RW_angle = [0 0 10 3];
            elseif and(eq(b4.tokens,1),lt(global_info.r1_LK_angle(1),0)),
                %Right Hand
                global_info.r1_RW_angle = [0 0 20 1];
            end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Taask '2' indicating Running motion
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    elseif eq(global_info.task(2),2),


        %For running
        %Left Legs

        %No of token in particular place
            b1 = get_place('pr1_LK_pos_al');
            b2 = get_place('pr1_LK_neg_al');


            if eq(b1.tokens,25),
                %$Right Hand
                global_info.r1_RE_angle = [5 0 0 2];
                global_info.r1_RW_angle = [0 0 120 2];
                global_info.r1_LK_angle = [-25 0 0 1];
                global_info.r1_LA_angle = [0 0 -35 8];
            elseif eq(b2.tokens,25),
                global_info.r1_RE_angle = [-30 0 0 2];
                global_info.r1_RW_angle = [0 0 90 2];
                global_info.r1_LK_angle = [25 0 0 1];
                global_info.r1_LA_angle = [0 0 -25 8];

            end



        %Right Legs

        %No of token in particular place
            b3 = get_place('pr1_RK_pos_al');
            b4 = get_place('pr1_RK_neg_al');


            if eq(b3.tokens,25),
                %$Right Hand
                global_info.r1_LE_angle = [5 0 0 2];
                global_info.r1_LW_angle = [0 0 120 2];
                global_info.r1_RK_angle = [-25 0 0 1];
                global_info.r1_RA_angle = [0 0 -35 8];
            elseif eq(b4.tokens,25),
                global_info.r1_LE_angle = [-30 0 0 2];
                global_info.r1_LW_angle = [0 0 90 2];
                global_info.r1_RK_angle = [25 0 0 1];
                global_info.r1_RA_angle = [0 0 -25 8];
            end

    end

    fire = 1;


end
```

## C3. Display Function

Slight modification was made in previously available function ‘*plotp.m*’; to have variable marker which represents different tokens in graph is displayed below.

**<u>plotpNew.m</u>**

```
function [TOKEN_MATRIX] = plotpNew(sim_results, set_of_places, ...
                            plotCOLOR, plotLINEWIDTH)
%        [TOKEN_MATRIX] = plotp(sim_results, set_of_places, ...
%                           plotCOLOR, plotLINEWIDTH)
%
% Purpose:  To plot simulation results;  to plot how tokens change with time
% Input parameters: Simulation Results (the structure output by ?gpensim?)
%                   {set_of_place_names}
%                   global_info (optional)
% Out parameters:   TOKEN_MATRIX (contains tokens of places with time)
% Uses: extractp  (extracts tokens from the SIM results structure)
% Used by:  [main simulation file]
% Example:
%   % in main simulation file
%   sim = gpensim(png, dynamic);
%   plotp(sim, {'p1','p2','p3'}, 'r', 10);

%  Reggie.Davidrajuh@uis.no (c) Version 6.0 (c) 10 july 2012

% Modified for varible markers by Bikee Maharjan : 10/06/2015
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global PN;
PN = sim_results;

TOKEN_MATRIX = extractp(set_of_places);
[nr_rows, nr_columns] = size(TOKEN_MATRIX);

time_series = TOKEN_MATRIX(2:nr_rows,1); %skip place indice
TOKENS = TOKEN_MATRIX(2:nr_rows,2:nr_columns); %ONLY TOKENS

xunits = 'Time'; % initially
if isfield(PN, 'HH_MM_SS'),
    if (PN.HH_MM_SS),
        if gt(PN.completion_time, 60*60),
            xunits = [xunits ' in HOURS'];
            time_series = time_series/(60*60);
        elseif gt(PN.completion_time, 60),
            xunits = [xunits ' in MINUTES'];
            time_series = time_series/60;
        else
            xunits = [xunits ' in SECS'];
        end; % if gt(sim_results.completion_time
    end; % (sim_results.HH_MM_SS),
end; % isfield(sim_results,HH_MM_SS)



p = plot(time_series, TOKENS,... % DEFAULT:: linewidth=.5, MarkerSize=10
     'linewidth', 0.5, 'MarkerSize', 5);

 % For variable markers for diffrent lines in same graph
  markers = {'h','o','*','^','>','s','d','.','v','x','<','p','+'};

  for i = 1:numel(p), %p is output of plot
```

```
    set(p(i), 'Marker', markers{i});
  end;

if ge(nargin, 3),
    if (plotCOLOR), set(p, 'color', plotCOLOR); end;
end;
if eq(nargin, 4), set(p, 'LineWidth', plotLINEWIDTH); end;

grid on; legend(set_of_places); xlabel(xunits); ylabel('Number of tokens');
```