Warsaw University of Technology

FACULTY OF
POWER AND AERONAUTICAL ENGINEERING

Institute of Aeronautics and Applied Mechanics

Master's diploma thesis

in the field of study Automatic Control and Robotics
and specialisation Robotics – EMARO

Investigation of the design method for developing the control system for walking machines

Razeen Hussain
student record book number 287703

thesis supervisor
Prof. dr hab. inż. Teresa Zielińska

co-supervisor
Dr Matteo Zoppi

Warsaw, 2017

Investigation of the design method for developing the control system for walking machines

Abstract:

Walking machines have proven to be an important invention as they do not require any prepared surface. This ability is specially vital when the robot has to navigate in an unexplored environment. The walking machines are equipped with a large number of actuators and sensors to achieve a robust locomotion. The complex nature of such machines requires a systematic approach to designing the control system. This research focuses on developing a functional control structure based on the logic labelled finite state automaton approach for walking machines. A general control structure is proposed and a hexapod walking machine is used to verify the practicability of the proposed design.

Keywords: walking machines, control system, real-time control, finite state machines, mobile robotics, behaviour models

Badanie metody projektowania dla syntezy systemu sterowania maszynami chodzącymi

Streszczenie:

Maszyny kroczące są ważnym narzędziem, ponieważ do przemieszczania przygotowanej powierzchni. Ta zdolność jest szczególnie istotna, gdy robot musi poruszać się w niezbadanym środowisku. Maszyny kroczące są wyposażone w dużą liczbę siłowników i czujników. Złożona charakter takich maszyn wymaga systematycznego podejścia do projektowania systemu sterowania. Badania koncentrują się na opracowaniu funkcjonalnej struktury sterowania opartej na podejściu automatu stanu skończonego oznaczonego logiką dla maszyn chodzących. Zaproponowana jest ogólna struktura sterowania, a do sprawdzenia praktyczności proponowanego systemu wykorzystuje się maszynę chodzącą heksapod.

Słowa kluczowe: maszyny kroczące, system sterowania, sterowanie w czasie rzeczywistym, skończone automaty, roboty mobilne, modele zachowań

Statement of the author of the thesis

Being aware of my legal responsibility, I certify that this diploma:

- has been written by me alone and does not contain any content obtained in a manner inconsistent with the applicable rules,
- had not been previously subject to the procedures for obtaining professional title or degree at the university

Furthermore I declare that this version of the diploma thesis is identical with the electronic version attached.

…… ……..……

date author's signature

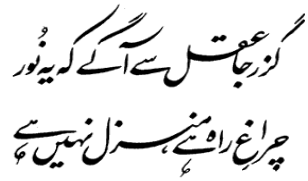
Statement

I agree to make my diploma thesis available to people, who may be interested in it. Access to the thesis will be possible in the premises of faculty library. The thesis availability acceptance does not imply the acceptance of making the copy of it in whole or in parts.
Regardless the lack of agreement the thesis can be viewed by:
- the authorities of Warsaw University of Technology
- Members of The Polish Accreditation Committee
- State officers and other persons entitled, under the relevant laws in force in the Polish Republic, to free access to materials protected by international copyright laws.

Lack of consent does not preclude the control of the thesis by anti-plagiarism system.

…… ……..……

date author's signature



“Go beyond the path of reason because reason can only be the light
That guides your path, it cannot be the goal of life”

– Muhammad Allama Iqbal

Contents

Acknowledgements **xv**

Summary **xvii**

1 Introduction **1**

1.1 State of the Art 2

1.1.1 Control System for the Walking Machines 2

1.1.2 Control Structure 3

1.1.3 Navigation 5

1.1.4 Real-time Control Systems 6

1.1.5 Examples of Control Systems for Walking Machines 6

1.2 Motivations 9

1.3 Objectives 9

1.4 Problem Statement 10

2 Finite State Machines **12**

2.1 Selection of FSM Tool 12

2.2 MiEditLLFSM Framework 13

2.3 Communication via Whiteboard 14

2.4 Formal Language for FSMs 15

3 Methodology **16**

3.1 The Control Structure 16

3.2 Sensory Reading 19

3.3 User Interface 20

3.4 Global Navigation 22
3.5 Local Navigation 23
3.5.1 Gait Selection 25
3.5.2 Real time path planning 26
3.5.3 Heading Correction 27
3.6 System Faults 28
3.7 Position Evaluation 28
3.8 Generation of Leg-end Trajectories 30
3.9 Comparison of the Approaches 31

4 Experimental Setup and FSM Implementation 34
4.1 The Simulator 34
4.2 Hexapod Walking Machine 35
4.2.1 Gait Motion 37
4.3 Proximity Sensors 38
4.4 Navigation Conditions 38
4.5 Interface 40
4.6 Environments 44
4.7 Implementation of the FSMs 45
4.8 Real-time Implementation 46

5 Experimental Results 47
5.1 Discussion 56

6 Conclusion 57

Bibliography 61

Appendix 65

List of Figures

1-1 Hierarchical control structure 4
1-2 Subsumption control architecture 5
1-3 Control structure of the GROVEN hexapod 7
1-4 Control structure of the Big Dog quadruped 8
1-5 Control structure of the Ambler hexapod 9

2-1 Basic elements of the MiEditLLFSM notation 14

3-1 Proposed control structure 17
3-2 Detailed control structure 18
3-3 Basic FSM model for the sensory reading subsystems 20
3-4 Activities of the user interface subsystem represented by a FSM . 21
3-5 Activities of the global navigation subsystem represented by a FSM 22
3-6 Activities of the local navigation subsystem represented by a FSM 24
3-7 Activities of the sub-behavioural system for the local navigation subsystem represented by a FSM 26
3-8 Demonstration of a bug algorithm for a preferred left turn robot . . 27
3-9 Heading correction principle handling the path deviation 27
3-10 Reference frames for the robot navigation 28
3-11 Transformations between the coordinate frames (roll, α; pitch, β; yaw, γ) 29
3-12 Process flow for the brain process of the GROVEN hexapod 32

4-1 The rectangular body architecture for hexapods 35
4-2 The leg structure of the hexapod 36

4-3 Simulation of the hexapod robot 36
4-4 Gait diagram for tripod gait 37
4-5 Gait diagram for wave gait 37
4-6 Proximity sensor schematic 38
4-7 Arrangement of the proximity sensors 39
4-8 Possible scenarios for the hexapod motion 39
4-9 The developed user interface 41
4-10 User interface along with the simulator view 42
4-11 Pop-up message for mission completion 43
4-12 Pop-up message for a locked path 43
4-13 Pop-up message for a system fault 43
4-14 Typical environment 44

5-1 Robot trajectory for an open path 48
5-2 Robot trajectory for a path with an obstacle 48
5-3 Robot trajectory for a path with an obstacle placed centrally 49
5-4 Robot trajectory for a path with a slanted obstacle 50
5-5 Robot trajectory for a path with an inclined surface 50
5-6 Robot trajectory for a path with an obstacle blocking leftward motion 51
5-7 Robot trajectory for a locked path 52
5-8 Robot trajectory for an open corridor path 52
5-9 Robot trajectory for a corridor path with obstacles 53
5-10 Robot trajectory for a blocked corridor path 54
5-11 Robot trajectory for an online modified mission 54
5-12 Modified user interface for the system faults 55
5-13 Robot trajectory for a mission suffering from a system fault 55

Acknowledgements

First of all, I would like to thank Allah Almighty for His blessings and giving me the strength to finish this research in time.

I am thankful to my thesis supervisor, Dr. Teresa Zielińska who has been a source of constant support throughout the project. Her guidance and encouragement proved invaluable for the success of this project.

I would also like to thank my thesis reviewer, Dr. Matteo Zoppi from University of Genoa for providing valuable comments on the thesis.

I am grateful to Dr René Hexel from Griffith University and Maksym Figat from Faculty of Electronics and Information Technology for sharing their technical expertise in the research area.

I would like to express my profound gratitude to all the people involved in the EMARO program. Without their support, this would not have been possible.

Last but not the least, I am thankful to my family for their moral support and prayers without which the research could not have been completed.

Summary

A logic labelled finite state automaton approach was presented to design a functional structure of a walking machine's control system. The control system was decomposed into three distinct functional subsystems arranged in a hierarchical structure where each subsystem's activities were defined by a FSM.

The MiEditLLFSM tool, an editor for logic labelled finite state machines, was used to model the behaviour of the FSMs. The tool allowed executable high level behaviour models to be created facilitating both the modelling of the activities of the control system and automatic code generation. The tool also manages the task scheduling for all the FSMs using a single sequential scheduler. Communication between the various FSMs was done using the whiteboard.

The proposed control system allows for a remote operator to set waypoints for the walking machine. The path cannot be pre-planned and is generated in real-time based on sensory information. The navigation algorithm implemented is a limited knowledge path planner.

The FSM based approach was tested on a V-REP simulation which used a hexapod robot as an example. Various sensors needed for the control system to function correctly were incorporated in the physical structure of the hexapod. The robot was made to navigate in different environments. Obstacles were placed in the robot's path to verify the obstacle avoidance algorithm.

The work done provides a basis for designing robotic control systems based on the logic labelled finite state automaton approach. As a next step, the implementation of the proposed approach can be extended to control an actual hexapod.

Chapter 1

Introduction

One of the most important outcome of biological evolution is the ability to walk since it does not require any prepared surface. Traditional mobile robots mostly use wheels which work well with only flat surfaces. For robots to be able to navigate in unknown environments, it is imperative for them to imitate the walking behaviour of biological species. Consequently, robotics has evolved to address this issue by coming up with the walking machines. The multi-legged walking machines do not require all appendages to be touching the ground making them more suitable for utilization in uneven environments.

Many types of walking machines have been developed. Bipedal robots imitate human beings, however, for a legged robot to be statically stable, it should comprise of three or more legs. For this reason, the quadrupeds and hexapods are the most common walking machines. These robots have large number of actuators and many sensors for their locomotion. It is evident due to their complex nature, that a systematic approach for designing the control system of such machines is crucial for its closed loop performance.

Mechanical design and motion generation for the multi-legged walking machines have been the main focus of researchers. Although, these are important aspects for the development of robots, it should be noted that the control system realization problems have been a hurdle in their successful practical implementations.

By studying the insect locomotion, it can be seen that they have different

gait cycles and switch between them based on feedback coming from sensory neurons. Similar behaviour can be associated with walking machines. The robot should be able to alter its gait cycle based on sensory feedback, adapting to the changing terrain. The gaits can be interpreted as different states of the robot locomotion, making it ideal for representation as a finite state machine (FSM).

The purpose of this research is to elaborate and test the functional structure of a control system using a Logic Labelled Finite State Automatons (LLFSMs) approach. The functional structure under consideration is being designed for a walking machine performing exploration tasks. The main focus will be to use a systematic approach to create a system having real-time capabilities.

For this, V-REP will be used as the simulator and MiEditLLFSM tool will be used to implement the finite state automatons. The implementation of the developed control structure will be done on a hexapod robot. Since the problem will be evaluated using simulations, the driver configuration will not be addressed in this research.

1.1 State of the Art

The research focuses on the implementation of a functional structure of a walking machine control system using a finite state automaton approach. For this purpose, several fields need to be analysed and discussed. First, the aspects of a robotic control system will be discussed. This will be followed by a description of some already developed control system structures.

1.1.1 Control System for the Walking Machines

The control system of any robot is a crucial component for its success. It dictates how the robot behaves. Usually robots exhibit reactive behaviour. Thus sensors play an important role for optimal closed loop control.

Typically, there are many sensors present in the system acting as the input for the control system. A global positioning system (GPS) allows the evaluation of

the starting position of the walking machine whereas a compass gives the heading direction (yaw angle) towards the goal. For detection of terrain conditions, the walking machine is equipped with inclinometers which calculate the roll and pitch angles. Proximity sensors allow detection of obstacles, digital encoders give feedback on the position of the leg joints while the leg ends are equipped with contact sensors as well.

The terrain inclination governs the gait of the walking machine. The control system must be capable of handling all the gaits and the sequence of the leg transfer motion from one periodic gait to another.

1.1.2 Control Structure

One of the most essential principle in designing a modern software architecture is modularity. The main system is divided into several smaller modules, each responsible for a specific task. This feature offers greater flexibility in the design and allows augmentation and exclusion of modules. Thus, allowing for a system developed for one robot to be used for another one simply by replacing a module.

The arrangement of these modules along with how they interact with each other determines the control structure of the robot. The two most common approaches to designing a robotic control structure are as follows:

Hierarchical Architecture:

The system is decomposed into several subsystems which are arranged in a top-down manner. Each subsystem carries out a functional task with the higher level tasks being more abstract. A typical example would be the upper layer performing motion planning while the lower level controlling the actuators. Natural systems are often modelled in this manner [1]. This approach is most popular with mobile robot applications [2–4] especially those pertaining to walking machines [5–7].

An example of a hierarchical architecture for a walking machine is shown in Fig. 1-1. The highest layer produces the body trajectory. The navigation

algorithm is implemented here. The next level generates the gaits. The third layer is responsible for producing the leg-end trajectories. The lowest level uses an inverse kinematics model to compute the joint configurations from the leg-end trajectories. A two way communication exists between the layers so that feedback is provided.

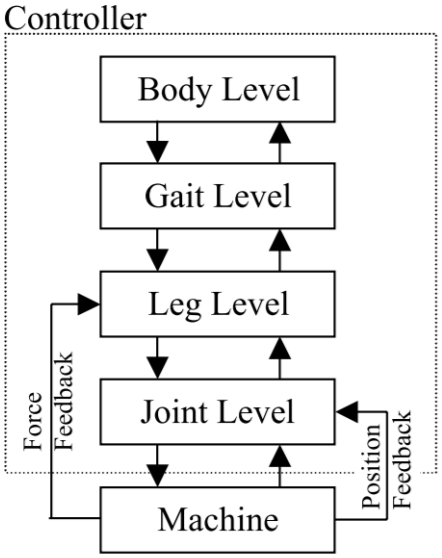


Figure 1-1: Hierarchical control structure [8]

Subsumption Architecture:

Subsumption architecture is a behavioural architecture. Each module in the control structure has an assigned priority. The modules are arranged in a hierarchical manner with the highest priority placed as the top most layer. The layers operate in parallel and use sensory information to produce outputs. However, a higher priority layer can subsume a lower priority module. This control architecture is popular with real-time control systems of robots navigating in dynamic or unknown environments [9, 10].

An example of a exploration robot exhibiting a subsumption architecture is shown in Fig. 1-2. It decomposes the system into various behaviours which are arranged in a hierarchy. The lowest layer allows the robot to avoid obstacles. When an obstacle approaches, the robot moves away from it. However, when

there is no obstacle in sight, the robot does nothing. The upper layer behaviour allows the robot to move around in its environment. Due to the lower layered *Avoid Obstacle* behaviour, the *Wander* behaviour can subsume the obstacle avoidance layer and combined together can avoid obstacles while wandering around. The same concept is applied to all the layers. All layers have access to the sensors and the actuators. There is a two-way communication between the behavioural layers so that each layer can support each other.

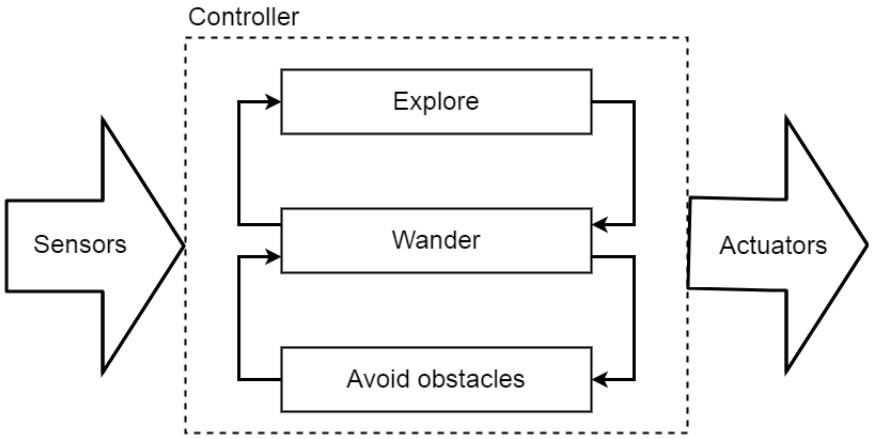


Figure 1-2: Subsumption control architecture

There are many other control structures. However, hierarchical systems are most popular for walking machines. Some examples of the walking machine's control system structures will be discussed in Sec. 1.1.5.

1.1.3 Navigation

The navigation from start to goal position is an important aspect for the exploration task. The most effective and shortest path from start to goal is a straight line connecting the two points. However, given an unknown environment, the robot must be capable of detecting obstacles and generating a path around the obstacle before continuing its original path. For this purpose, the trajectory cannot be pre-determined and it needs to be updated after each motion.

Typically, legged robots can only turn at discrete angles. This can either cause the robot to never point towards the goal position or the robot may have to

correct its heading direction after each movement. To avoid this predicament, a $\pm 10^{\circ}$ heading error is generally permissible in the system.

1.1.4 Real-time Control Systems

Real-time systems need to react to external events within specific time constraints [11]. Thus the correctness of such systems depends not only on producing the adequate response but also on temporal factors i.e. when the response is produced.

In a real-time operating system (RTOS), tasks are executed using threads. Multiple threads run concurrently within a process and share the memory. A scheduler is required to manage the execution of these threads. At each time slice, a certain thread gets access to the processor's resources such as memory.

For the control system to work effectively, it is essential that it possesses real-time properties. This is because there are many actuators being controlled in parallel and many sensors to be served. The concurrent processes require data exchange for real-time control, this communication needs to be synchronized against time slices. Also, each process should wait for the previous action to be completed before execution of the next command.

The RTOS are usually incorporated into the embedded system with regards to robotic applications. Over the years, many real-time operating systems have been developed such as VxWorks [12], QNX [13] and RTAI [12]. QNX is considered the best RTOS as it offers fast and predictable performance and has an excellent architecture for implementing a robust distributed system.

1.1.5 Examples of Control Systems for Walking Machines

A functional structure of control system has been developed for hexapods [14, 15]. The hexapod considered in those works had 18 DOF so there are 18 motors being operated synchronously to generate the gait. The control system structure shown in Fig. 1-3 consists of three main processes, namely the Brain, the Leg and the Driver.

- The brain process takes care of the global navigation for the hexapod. It communicates with the control station through radio communication. It gathers the GPS, compass and inclinometer data, calculates the current position and the distance to goal and heading towards the final position.
- The leg process is responsible for the local navigation of the hexapod. It handles the contact and proximity sensor data, generates the detailed path and gait evaluation based on the demand sent by the brain process. It is also capable of online modification of the leg trajectories. Obstacle avoidance is incorporated in this process as well.
- The driver process is the lowest level controller and is basically responsible for controlling the motors.

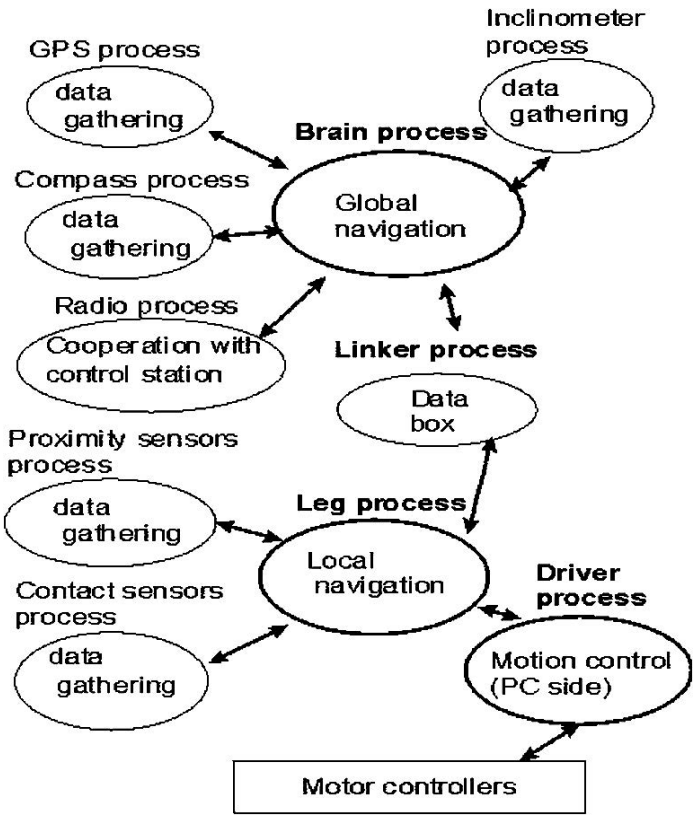


Figure 1-3: Control structure of the GROVEN hexapod [15]

The different processes were implemented on a QNX based embedded system using the Watcom C programming language. A client-server mechanism was used for the inter-process communication.

Similar control structures have been observed in other already developed

walking machines. The control structure of the Big Dog [16] quadruped developed by Boston Dynamics is shown in Fig. 1-4. It has a two layered structure. The top layer generates the trajectory while the lower layer configures the motor joints to fulfil the demand from upper level planning module. In addition to these two main subsystem, there are additional upper level modules that assist the trajectory planning. These include pose estimation and object tracking subsystems. A PC104 stack on-board computer equipped with a QNX operating system was used to implement the main control modules.

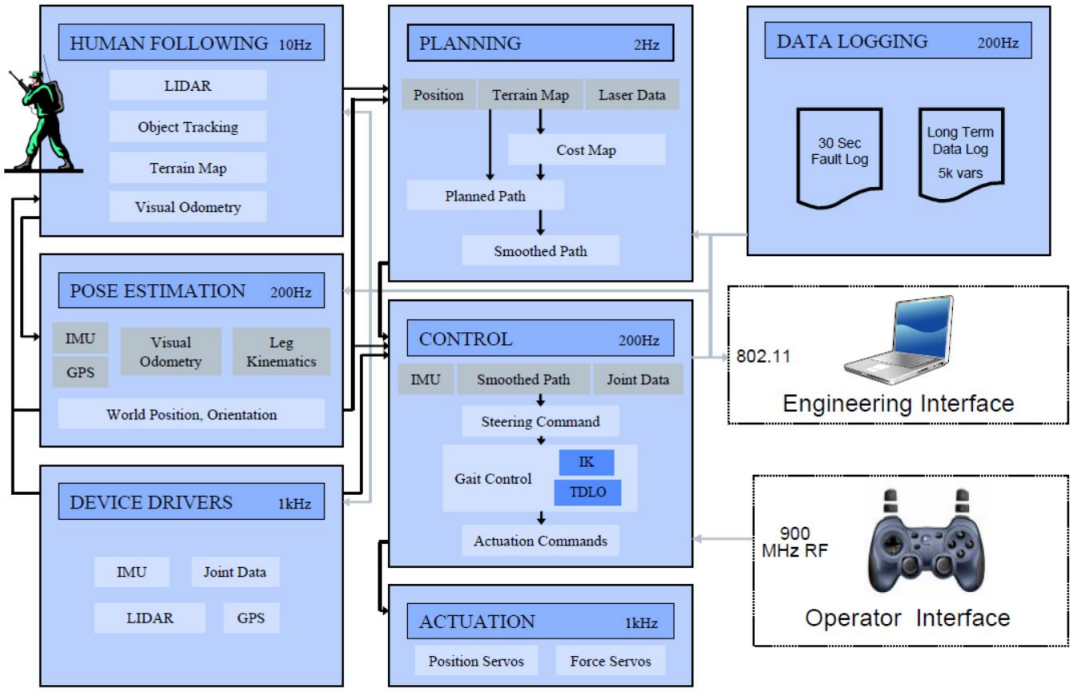


Figure 1-4: Control structure of the Big Dog quadruped [16]

Another example of a control system is that of the Ambler hexapod [17]. It uses a behaviour based approach. The robot activities are treated as behaviours which interact with each other through the environment. In addition to the behaviour based control structure, it makes use of a Task Control Architecture (TCA) to allocate resources to the individual behaviours. A Solaris based operating system was used along with a SUN workstation to implement the central and reactive control modules. The detailed control structure of the Ambler robot can be seen in Fig. 1-5.

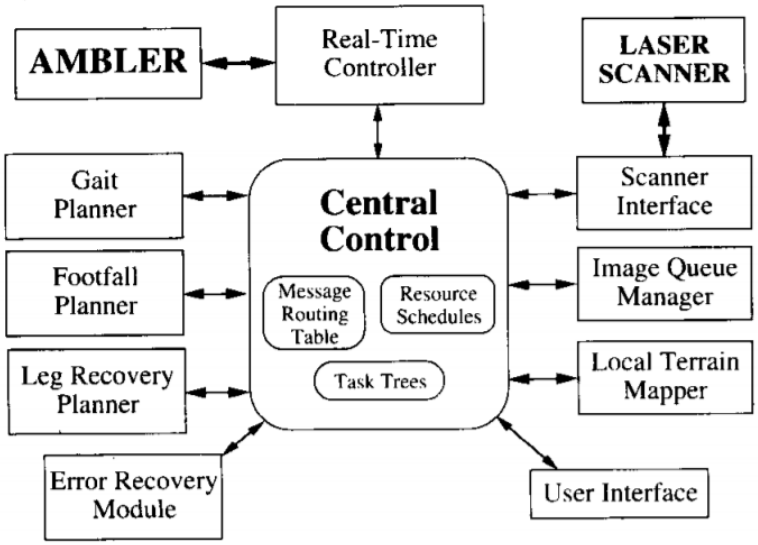


Figure 1-5: Control structure of the Ambler hexapod [17]

1.2 Motivations

Although many robotic control systems have been developed with real-time capabilities for the walking machines, the FSM methods were not yet applied. The FSM approach offers a more reliable, robust and quicker design solution. It is a promising recent tool that has demonstrated to be an effective technology for modelling the control systems in that, that of walking machines.

The research will aim to investigate the performance of a hexapod robot control system represented with a LLFSM approach with focus to its application on rough terrains for autonomous locomotion. Once the described LLFSM based control system has been developed, it can be further extended to accommodate path planning algorithms and later implemented on various walking machines available in the laboratory.

1.3 Objectives

The research will concentrate on designing a LLFSM based control system for walking machines with the implementation focused on a hexapod robot. The main objectives for this research are as follows:

- to propose motion models based on the study of insect locomotion;
- to model various motion behaviours of the hexapod as different states for the FSM framework;
- to design and implement the functional structure of the control system using the logic labelled finite state automaton approach;
- to develop a V-rep simulation for evaluating the developed model;

The developed system will have real-time capabilities and the modular approach will allow it to be implemented on any walking machine.

1.4 Problem Statement

The problem of designing a novel control structure for a walking machine will be addressed in this research. A detailed analysis of already developed systems revealed that most of them fail to produce satisfactory real-time capabilities while others had large complicated neural structures which are difficult to program and modify when used with a different walking machine.

The most important property of a legged machine is its ability to move on rugged terrains. It can adapt to various loads and surface types by modifying its gait cycle. On flat surfaces, the tripod gait is most effective as it offers faster speeds while maintaining stability. On steep terrains, the wave gait offers better support. When carrying load, a walking machine performs best with maximum leg support motion due to better weight distribution among the supporting legs. The control system should be able to handle the gait switches based on data provided by sensors.

Another vital aspect that needs to be incorporated into the control system is the walking machine's navigation from a start to a goal position. It should be able to localize in its unknown environment and move around obstacles if present in its desired path.

The control system under development, as discussed in the previous sections, will be based on the logic labelled finite state automaton approach. The various gait motions will act as different states of the machine. The information

coming from inclinometers and proximity sensors will dictate the transitions between these states.

The main problem is to ensure that the control system is able to handle real-time data and react appropriately. In an unknown rugged environment, the slope of the surface is constantly changing. The hexapod needs to adapt accordingly in sufficient time to ensure it does not topple down.

Chapter 2

Finite State Machines

Finite state automata/machine (FSM), a tool in model driven engineering (MDE), is a computational model based on a system comprising of a finite number of states. At any given time, only one state can be active. The machine can transition to another state based on logical statements that need to be proved by an inference engine.

Typical applications of the FSMs include speech recognition, vending machines, traffic lights, microwave ovens, video games and many more. Recently, FSMs have been shown to be a promising approach in modelling the behaviour of robots [18].

2.1 Selection of FSM Tool

Event driven approaches in which the transitions are labelled by events, are more popular for modelling the actions of a FSM. Over the years, many frameworks [19–21] have been developed for the implementation of such models.

However, the event-driven based models require implementation as a set of concurrent threads which communicate with each other. This multi-threaded approach increases the load on the system as a supplementary process must manage the synchronization of the threads and make sure there is no deadlock or thread starvation. A further drawback of the event driven models is that the verification process may result in a combinational explosion as all combinations

of the FSM states need to be correct. This considerably compromises the formal verification of the models both in the time domain and the value domain.

In order to resolve these issues, alternative approaches [22, 23] have been proposed. Such approaches support single thread execution of multiple FSMs. MiEditLLFSM tool is one of the frameworks that use this approach. It follows the approach of Harel's statecharts [24] and defines the behaviours as a mathematical model [25]. In contrast to the asynchronous event driven frameworks, this tool ensures synchronous FSMs. The MiEditLLFSM tool will be used to model the behaviour of the walking machines as it provides all the necessary tools for modelling.

2.2 MiEditLLFSM Framework

The MiEditLLFSM tool has been developed by MiPal. It represents a FSM as a graph, the states as nodes and transitions as arcs. The nodes are connected to each other by directed arcs. The arcs are labelled by boolean expressions. This is in contrast to the event driven approach where the arcs are labelled by events. The generality is not compromised [26] as the boolean expressions can also represent an event occurrence.

Each state of an FSM is attributed with three sections [27] which are described below:

- An *OnEntry* section contains the action that needs to be carried out when the system enters that state.
- An *OnExit* section contains the action that is executed when the system leaves the state and transition to another state.
- An *Internal* section contains the actions that are carried out when no transition is triggered. Unlike the other two sections which are executed only once, these actions are executed repeatedly until a transition condition becomes true.

Whenever a transition is enabled, the FSM switches to the next state and executes the *OnEntry* section of the new state. Each section only executes a

single action. Fig. 2-1 shows the basic elements in the MiEditLLFSM framework.

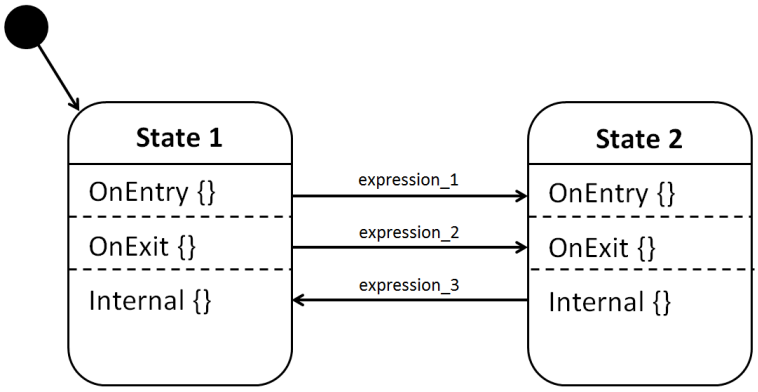


Figure 2-1: Basic elements of the MiEditLLFSM notation [22]

Multiple FSMs need to be executed simultaneously to ensure real-time capability in the system. The MiEditLLFSM scheduler ensures that at each time slice, one step of the FSM is executed satisfying concurrency in the system.

2.3 Communication via Whiteboard

The whiteboard [28] is a structured global database. It mimics the blackboard architecture [29]. It stores all the variables contained in the FSMs. However, at any time only one FSM should have access to the whiteboard, in order to provide lock-free atomicity. The FSM scheduler should ensure this.

There can be three types of variables in the system:

- *Local* variables are available only within a FSM.
- *Internal* variables are shared among the various FSMs in the system.
- *External* variables are outside the system. Typical example includes data from the sensors.

Whenever a section of the current state executes, it should first read the whiteboard and make local copies of the whiteboard variables. This is required so that all data being used by the action or transition function is recent.

2.4 Formal Language for FSMs

An FSM is defined by a set of states S and a transition function $T : S \times E \rightarrow S$ where E is a set of boolean expressions that govern the transitions [22]. There is always a state $s_0 \in S$, which is the initial state.

However in the MiEditLLFSM tool, T projects to a sequence instead. This is defined as $T(s_m, e_t) = s_n$ which means that the FSM will transit to state s_n when expression e_t holds true. The transition function when treated as a sequence lowers the burden on the behaviour designer.

Chapter 3

Methodology

The research aims to develop a control structure for walking machines based on the logic labelled finite state automatons approach. The approach as discussed previously decomposes the control actions into a set of different states. The machine can transition between these states based on logical conditions.

The control structure developed will be discussed first followed by a detailed explanation of the various automatons required for the robot to function properly. The proposed control structure can be adapted to any walking machine.

3.1 The Control Structure

A typical walking machine consists of many actuators that need to be controlled simultaneously. Furthermore, they are equipped with a lot of sensors for feedback to ensure optimal performance. This makes the real-time realization strenuous. Real-time control utilizes tasks distributed over concurrent processes.

The proposed control structure divides the overall system into smaller subsystems which are arranged in a hierarchical (top-down) structure. The control structure shown in Fig. 3-1 is used. Such a hierarchical control scheme was chosen as the developed control structure is not for a specific walking machine but can be used with any walking machine regardless of the number of actuators and sensors attached to it. A subsystem can be replaced by another one as suited for the walking machine. This incorporates modularity in the system. The modular

flexibility of the control structure allows it to be easily distributed over a network of micro-processors.

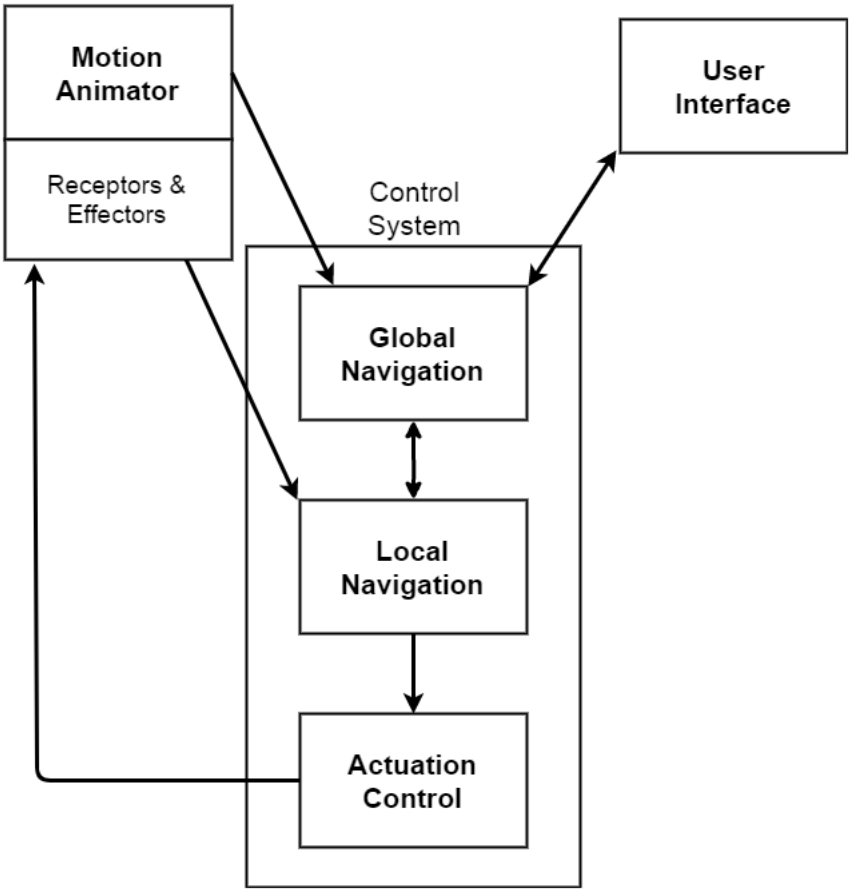


Figure 3-1: Proposed control structure

The highest level process sends the task demand to the lower level which in turn sends its demand to the lowest level. All processes are self-reliant and equipped with tools to meet the demand. However, only when the demand cannot be met, the higher level is informed and a new demand is requested.

Fig. 3-2 illustrates the detailed breakdown of the control system processes. The main tasks associated with each process are also mentioned. Each process in the control structure is treated as a separate finite state automaton. The data communication between automatons is done using the CL Whiteboard as described in Sec. 2.3.

Since the implementation will be done on a simulator running on a PC, the task scheduling of the concurrent processes involved need to be addressed. The

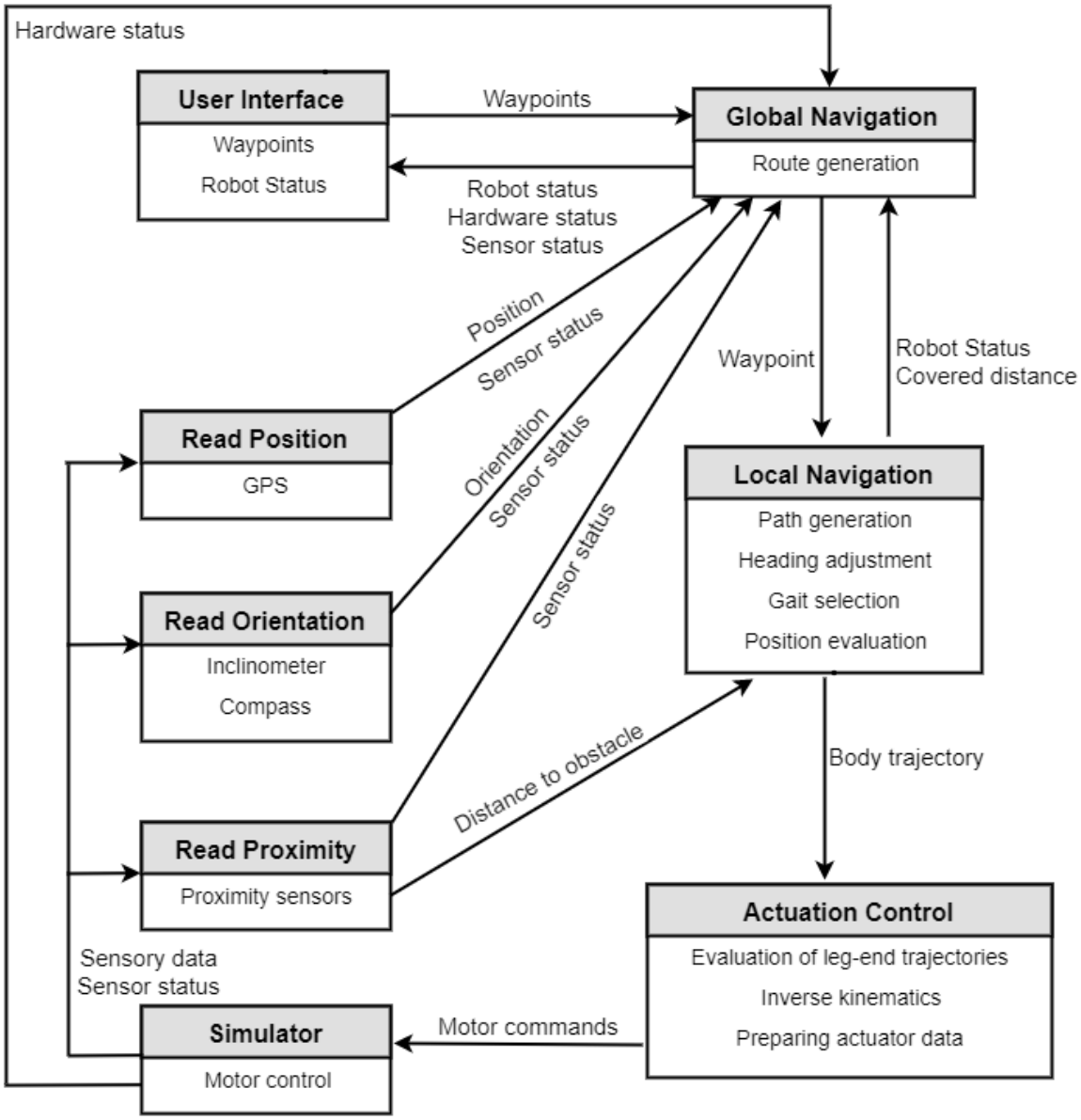


Figure 3-2: Detailed control structure

MiEditLLFSM framework that will be used to implement these finite state machines has an inbuilt concurrency model that takes care of the real-time task scheduling of multiple machines running simultaneously on a processor.

3.2 Sensory Reading

Walking machines are often equipped with numerous sensors to ensure adequate feedback required for its control. In a real-time system, all such sensors are treated as separate processes.

In our system, five types of sensors have been considered:

- **GPS:** Global Positioning System (GPS) is a popular method to determine the geolocation of an object. However, they typically have an accuracy of 3-6 meters. This amount of error is not acceptable in most applications. So, the information from the GPS sensor is used only to evaluate the starting position.
- **Inclinometer:** Inclinometers are devices that measure the slope of an object. These sensors are necessary to determine which gait cycle is more appropriate for a stable motion.
- **Compass:** Compass provides the heading direction of a robot. This information is necessary to ensure that the robot is moving in the correct direction.
- **Proximity sensors:** Proximity sensors are able to detect the presence of obstacles in the environment without physically interacting with the obstacle. These sensors are fundamental for obstacle avoidance; ensuring a collision free path to be taken by the robot.
- **Encoders:** Encoders help evaluate the angular position of motor shafts. They are essential for generating motor commands by ensuring the leg-ends are correctly moved and placed.

Each sensory process (*Read Position, Read Orientation, Read Proximity*) can be modelled as a single state machine. It reads the information from the physical sensor at a predetermined frequency and deposits the sensor data in

the sensor repository. Furthermore, it keeps track of any system failure and ends the process in case of a fault. The basic automaton developed to model the behaviour of the sensory reading processes is shown in Fig. 3-3. It should be noted that for each sensor the robot is equipped with, a similar independent FSM is required.

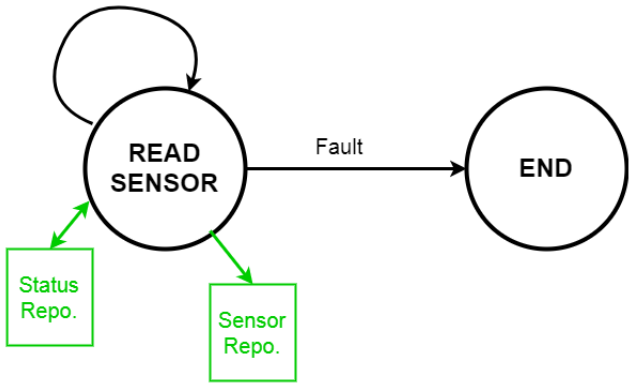


Figure 3-3: Basic FSM model for the sensory reading subsystems

A description of each state along with its transitions is explained below:

- **READ SENSOR:** In this state, the data is read from the physical sensor and deposited to the sensor repository at regular intervals. Also, a check is made at each cycle to ensure the sensor is operating properly. In case of a sensor failure, a fault message is generated and communicated to the other processes via the status repository.
- **END:** In case of a system failure, the sensory process transitions to the *END* state which terminates the process.

3.3 User Interface

The user interface is a channel for the operator and the control system to interact with each other. The operator cannot see the robot so the user interface needs to cater for this deficiency. It basically serves the following purposes:

- to acquire waypoints from the operator;
- to display the status of the robot;
- to show the trajectory of the robot;

- to caution the operator in case a fault occurs in the system;
- to inform the operator about a locked path and subsequently allow the operator to either stop the mission or change the waypoints;
- to abort a mission.

The graphical interface should be equipped with sufficient tools/control buttons to allow the remote operator to specify its demands. The robot status needs to be presented in a user-friendly manner.

There needs to be a user interface automaton in the system to assist the interaction between the user interface and the global navigation automaton. The user interface FSM can be modelled by a single state machine as depicted in Fig. 3-4.

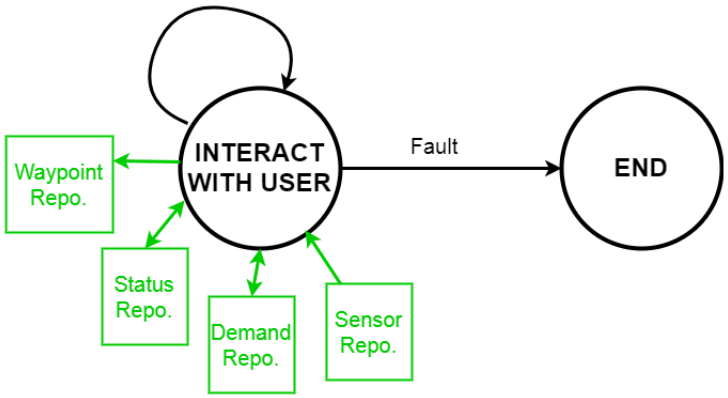


Figure 3-4: Activities of the user interface subsystem represented by a FSM

The states of this automaton are described below:

- **INTERACT WITH USER:** In this state, the user instructions are handled. When a new mission is sent by the user or the current one is modified, this state writes to the waypoint repository. Whenever, the user requests the robot status, it reads from the sensor and status repositories and communicates the information to the user. Stop mission request is also handled.
- **END:** When a system failure occurs, the mission needs to be aborted and communicated to the user. This state serves that purpose.

3.4 Global Navigation

The global navigation automaton is the highest level subsystem in the control structure and is responsible for route generation. It interacts with the user interface automaton; receiving target waypoints and requesting further instructions in case of a locked path. Moreover, the global navigation process monitors the mission; calculating the heading error and evaluating the position errors. This information is deposited on the status repository which is subsequently used by the local navigation subsystem.

Fig. 3-5 shows the activities carried out by the global navigation subsystem as described by a logic labelled FSM.

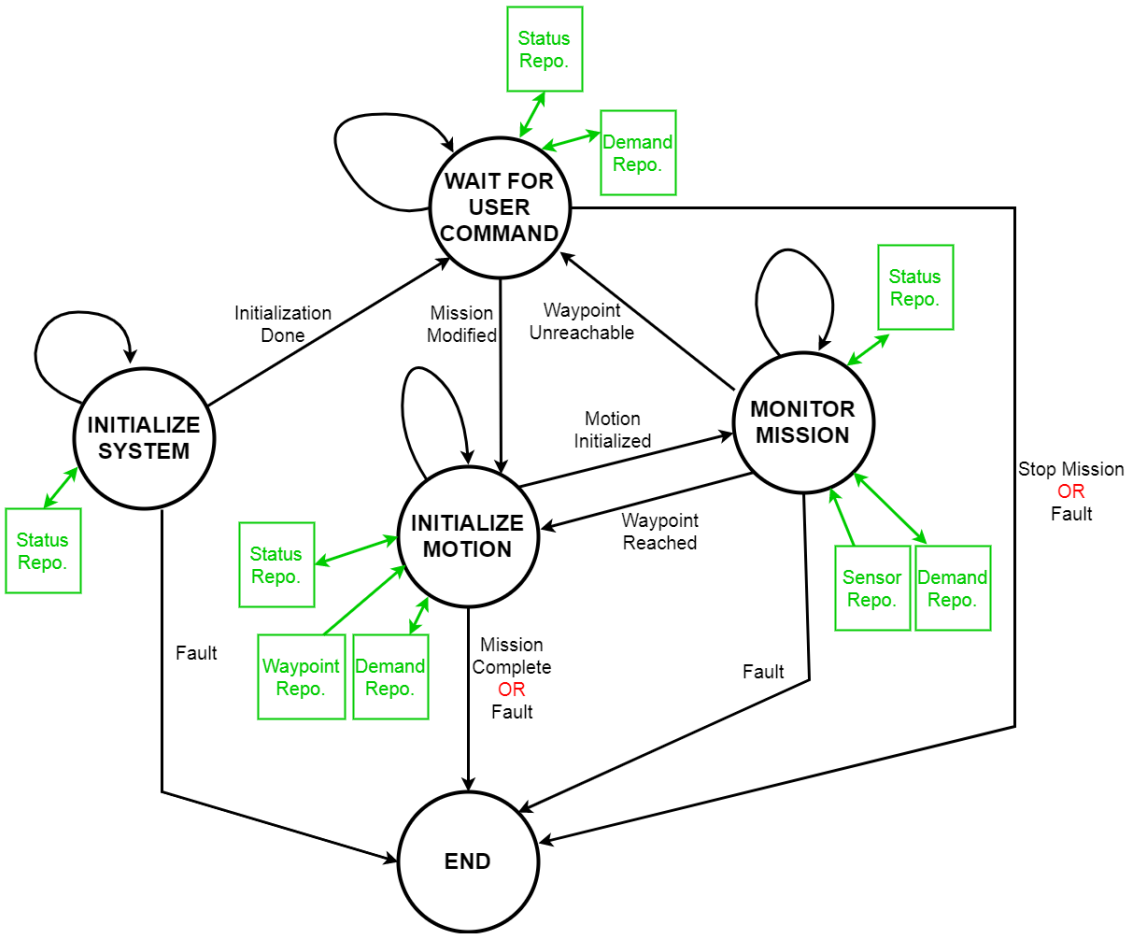


Figure 3-5: Activities of the global navigation subsystem represented by a FSM

Each state of the global navigation automaton is described below:

- **INITIALIZE SYSTEM:** This is the initial state of the automaton. It sets up the

repositories and performs a check that the robot system is working properly.

- **WAIT FOR USER COMMAND:** When the automaton enters this state, requests are sent to the user. These requests include request for a new mission (if there is no active mission available) or a request to modify or stop the mission in case of a locked path.
- **INITIALIZE MOTION:** This state reads from the waypoint repository and communicates the next goal position to the local navigation automaton and then transitions to the *MONITOR MISSION* state.
- **MONITOR MISSION:** While the local navigation automaton is navigating the robot through the unknown environment, this state prepares the basic status messages (position error, heading error, etc.) for the other processes. It uses data from the sensor repositories to calculate the various error in the system.
- **END:** In case of a system failure or a stop mission request from the user, the automaton transitions to this state which terminates the process.

3.5 Local Navigation

The local navigation subsystem is the middle level functional process in the hierarchy. It is responsible for detailed path navigation and gait selection. It uses information from the proximity sensors to produce a collision free path for the robot; fulfilling the route demand received from the global navigation subsystem. The information from the inclinometers is used to define the type of gait motion.

The navigation algorithm defines the body trajectory. This demand is sent to the lowest level in the hierarchy i.e. actuation control subsystem which generates the motor commands. Based on the body trajectory, the control system should also be able to evaluate the new position of the robot on its own since the GPS data is unreliable . The navigation algorithm and the position evaluation are explained in Sec. 3.5.2 and 3.7 respectively.

Fig. 3-6 shows the activities carried out by the local navigation subsystem as described by a logic labelled FSM.

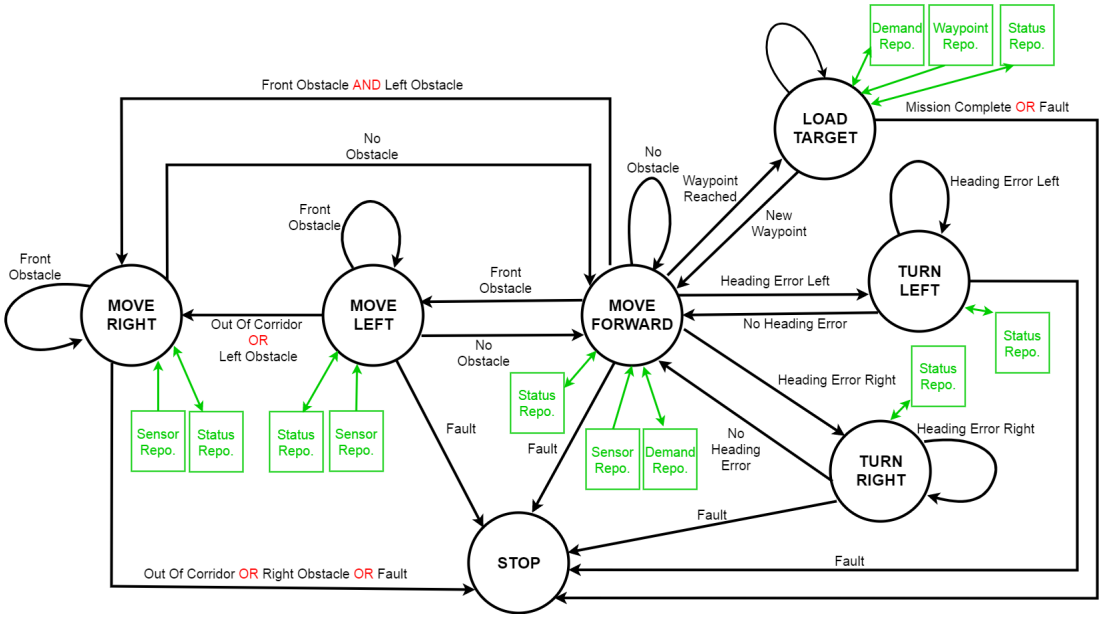


Figure 3-6: Activities of the local navigation subsystem represented by a FSM

The various states in which the local navigation automaton can transition to are described below:

- **LOAD TARGET:** This is the initial state of the automaton. It sends a request to the global navigation automaton that the current goal position has been reached and therefore requests a new target point. When a target waypoint has been loaded, it transitions to the *MOVE FORWARD* state.
- **MOVE FORWARD:** This state generates the body trajectory for a straight line motion. It also keeps track of obstacles present in the straight line motion by reading the sensor repository and transitions to the *MOVE LEFT* state in case an obstacle is detected in the forward direction. The heading errors deposited in the status repository by the global navigation automaton are used to perform heading corrections when necessary.
- **MOVE LEFT:** The gait motion for leftward motion is generated in this state. When the forward path clears, it transitions back to the *MOVE FORWARD* state. It also keeps track of the robot position within the corridor using a step counter to ensure the robot navigates in its defined proximities.
- **MOVE RIGHT:** When there is no path around the obstacle from the left side, the automaton transitions to this state which generates the rightward

gait motion.

- **TURN LEFT:** Whenever there is a requirement to perform heading correction with a left turning motion, the automaton transitions to this state.
- **TURN RIGHT:** Similarly to the *TURN LEFT* state, when a heading correction is required (but with a right turn motion), the automaton transitions to this state.
- **STOP:** In case of a system fault or the mission has been completed, the motion of the robot needs to be stopped. This state ensures the robot has stopped and returns the robot to its initial configuration.

3.5.1 Gait Selection

The local navigation subsystem is also responsible for the selection of gait. Two gaits are considered in this control system, namely the tripod gait and the wave gait. An overview of these gaits is presented in Sec. 4.2.1.

A two-tier definition of the behaviours is used. The upper level (the local navigation FSM as shown in Fig. 3-6) is a FSM which switches between one or more of these behaviours. The lower layer governs which type of gait to use. The two-tier description follows the concept of hierarchical FSMs. The generic template of the sub-behavioural FSM modelling the behaviours of the local navigation subsystem is demonstrated in Fig. 3-7. This arrangement of hierarchical FSMs is used with the *MOVE FORWARD*, *MOVE RIGHT* and *MOVE LEFT* behaviours.

The various states of the sub-behavioural FSM for each motion state of the local navigation subsystem are are described below:

- **TRIPOD GAIT:** This state generates the tripod gait. The FSM transitions to this state when the motion surface is relatively flat.
- **WAVE GAIT:** The wave gait is implemented in this state. Since this is the most stable gait, the FSM transitions to this state when a surface inclination is detected.
- **END:** In case of a fault, the sub-behavioural FSMs transition to this state, subsequently terminating the process.

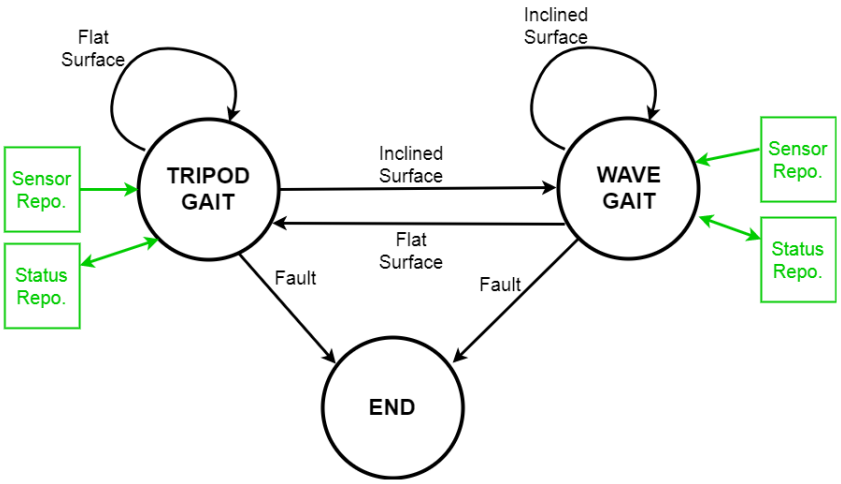


Figure 3-7: Activities of the sub-behavioural system for the local navigation sub-system represented by a FSM

3.5.2 Real time path planning

The target positions or waypoints are specified by a remote operator using the user interface. The operator is unaware of the presence of obstacles in the environment and the terrain conditions so the path the robot takes can not be pre-determined. It is necessary for the robot to be equipped with necessary tools that allow for an online generation of trajectory from one waypoint to another.

The precise trajectory taken by the robot is dependant on the environment conditions. With the help of proximity sensors, the robot navigates. A safe distance from obstacles must be kept throughout the mission.

The local navigation automaton incorporates the bug algorithm [30] which is an insect inspired navigation algorithm to navigate in unknown environments. This algorithm allows the robot to take the shortest possible route between the waypoints. The robot initially corrects its heading direction to point towards the goal and then starts moving in that direction. When an obstacle is encountered, it tries to move around it until the path is free again. Fig. 3-8 demonstrates the working of a typical bug algorithm. The basic steps involved are:

1. move towards goal position
2. follow obstacle until the path is free
3. continue moving towards the goal position

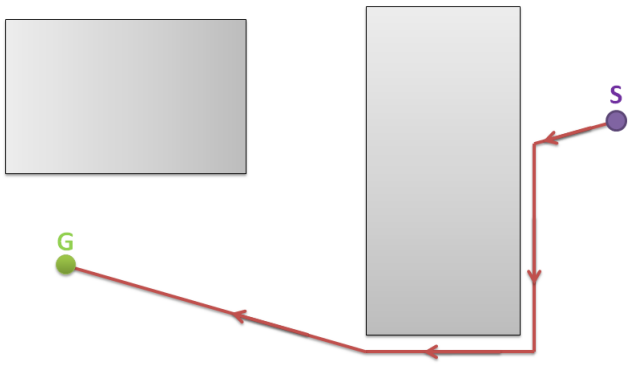


Figure 3-8: Demonstration of a bug algorithm [30] for a preferred left turn robot

3.5.3 Heading Correction

During the motion of the walking machine, there is a possibility that the robot may deviate from its original path. This could be due to many reasons such as surface slippage, initial error, external factors to name a few.

It is important that the control system makes sure that the robot is always moving towards the goal. However, correcting its heading with every angle of deflection is impractical. Thus, a tolerance of $\pm 5^{\circ}$ in the angle α is incorporated in the system. The principle is demonstrated in Fig. 3-9.

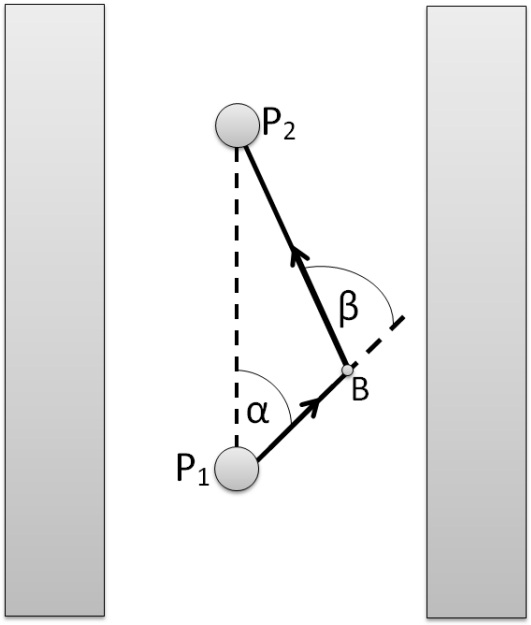


Figure 3-9: Heading correction principle handling the path deviation [15]

3.6 System Faults

As mentioned in the above automatons, the control system needs to take into account system failures as well. Typical faults considered are:

- **Sensor failure:** This includes damage to the physical sensor.
- **Embedded system fault:** This includes damage to the actual embedded system.
- **Data corruption:** Data may be corrupted when it is transmitted between various subsystems.

3.7 Position Evaluation

The robot is equipped with a GPS to help evaluate the position. However, due to a low accuracy, the data from the GPS is not reliable. So, the robot control system should be able to evaluate its own position by other means.

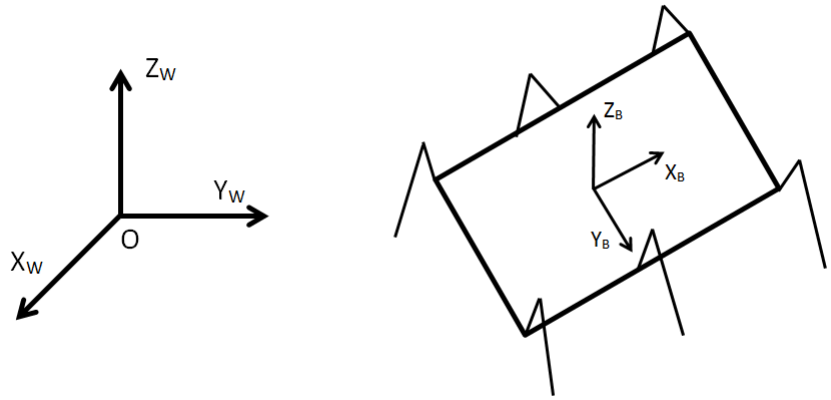


Figure 3-10: Reference frames for the robot navigation [15]

Fig. 3-10 shows a body frame B attached to the centre of the walking machine and a world frame W representing the global reference frame. The roll pitch yaw rotations between the two frames is presented in Fig. 3-11. The walking machine's body frame B can be expressed in the world frame W by the homogeneous transformation matrix ${}^{W}_{B}T$. The transformation matrix is defined by Eq. 3.1 where $s\alpha = \sin\alpha$, $c\beta = \cos\beta$, etc.

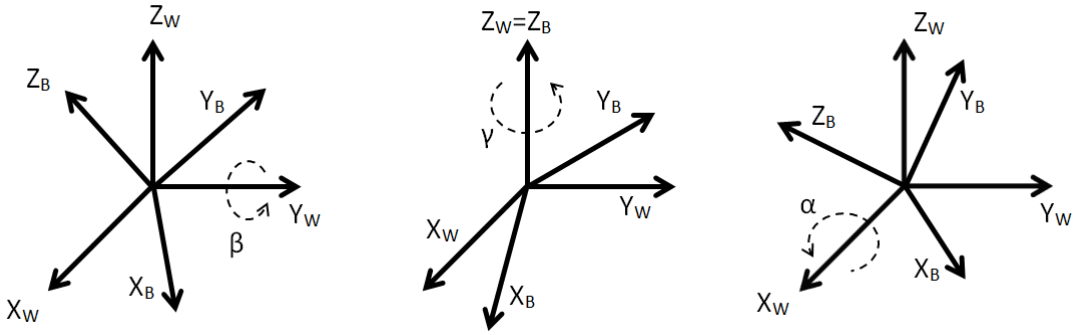


Figure 3-11: Transformations between the coordinate frames (roll, α; pitch, β; yaw, γ) [15]

$$
{}^{W}_{B}T = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta c\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma - s\alpha s\gamma & p_x \\ s\alpha c\gamma & s\alpha s\beta s\gamma - c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & p_y \\ -s\beta & c\beta s\gamma & c\beta c\gamma & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}
$$

The γ angle is the heading direction measured by the compass while α and β are the surface inclination angles and are measured by the inclinometers. The first three columns are the rotation matrix ${}^{W}_{B}R$ (Eq. 3.2) while the last column is the translation, P (Eq. 3.3), between the body frame B and the world frame W. This position vector defines the absolute position of the walking machine.

$$
{}^{W}_{B}R = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta c\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma - s\alpha s\gamma \\ s\alpha c\gamma & s\alpha s\beta s\gamma - c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \tag{3.2}
$$

$$P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{3.3}$$

The localization of the robot can be done using an incremental approach. At each cycle of motion, the robot moves forward a distance referred to as the stride length S_l. Using Eq. 3.4, the new position of the robot can be calculated.

$$P_{n+1} = P_n + {}^{W}_{B}RS_l \tag{3.4}$$

3.8 Generation of Leg-end Trajectories

The applied simulator does not require the leg-end trajectory generation, however, for completeness of description some information is provided on it. Actuation control is the lowest level in the control hierarchy. It receives the body trajectory from the local navigation subsystem and evaluates the leg-end trajectories. Once the leg-end trajectory is determined, it is fed into the inverse kinematics module which determines the motor commands.

The leg-end trajectories are generated in the body frame B. For a n-legged walking machine, the k^{th} leg's coordinates can be expressed in the body frame B by Eq. 3.5. Thus, the leg-end pose can be conveniently calculated with the Eq. 3.6.

$${}^{B}X_k = \begin{bmatrix} {}^{B}x_k \\ {}^{B}y_k \\ {}^{B}z_k \\ 1 \end{bmatrix} \tag{3.5}$$

$$^{W}X_k = {}^{W}_{B}T^{B}X_k \tag{3.6}$$

The inverse kinematics depends on the leg structure and is beyond the scope of this research.

3.9 Comparison of the Approaches

The control structure of the GROVEN hexapod mentioned in Sec. 1.1.5 will be used for comparison as the proposed control system is based on it.

There are some similarities in the two control system structures. Both use a hierarchical architecture. They divide the overall system into a series of smaller functional subsystems which are arranged in a hierarchical manner. The top layer sends its demands to the middle level which is equipped with all the necessary tools to meet the demand. The middle layer sends the body trajectory demands to the lowest level. The motor control layer then tries to fulfil the demand by transforming the body trajectory to leg-end trajectories and subsequently, generating the motor commands.

There is no central controller. All layers in the hierarchy are capable of functioning independently. They monitor the sensor values on their own and only require the demand instructions from the upper level subsystem.

There are considerable differences in the implementation of the individual subsystems. While the GROVEN hexapod control system uses more or less a sequential approach in implementing the various layered subsystems, the proposed control system makes use of hierarchical FSMs instead.

Fig. 3-12 shows the process flow of the global navigation module used by GROVEN hexapod. It waits for the communication to the radio process to be established. Then it requests target points and sends this information to the lower level subsystem. In a loop, it keeps check of the position of the robot and notifies the user when the demand has been met.

In contrast, the global navigation subsystem for the proposed control system

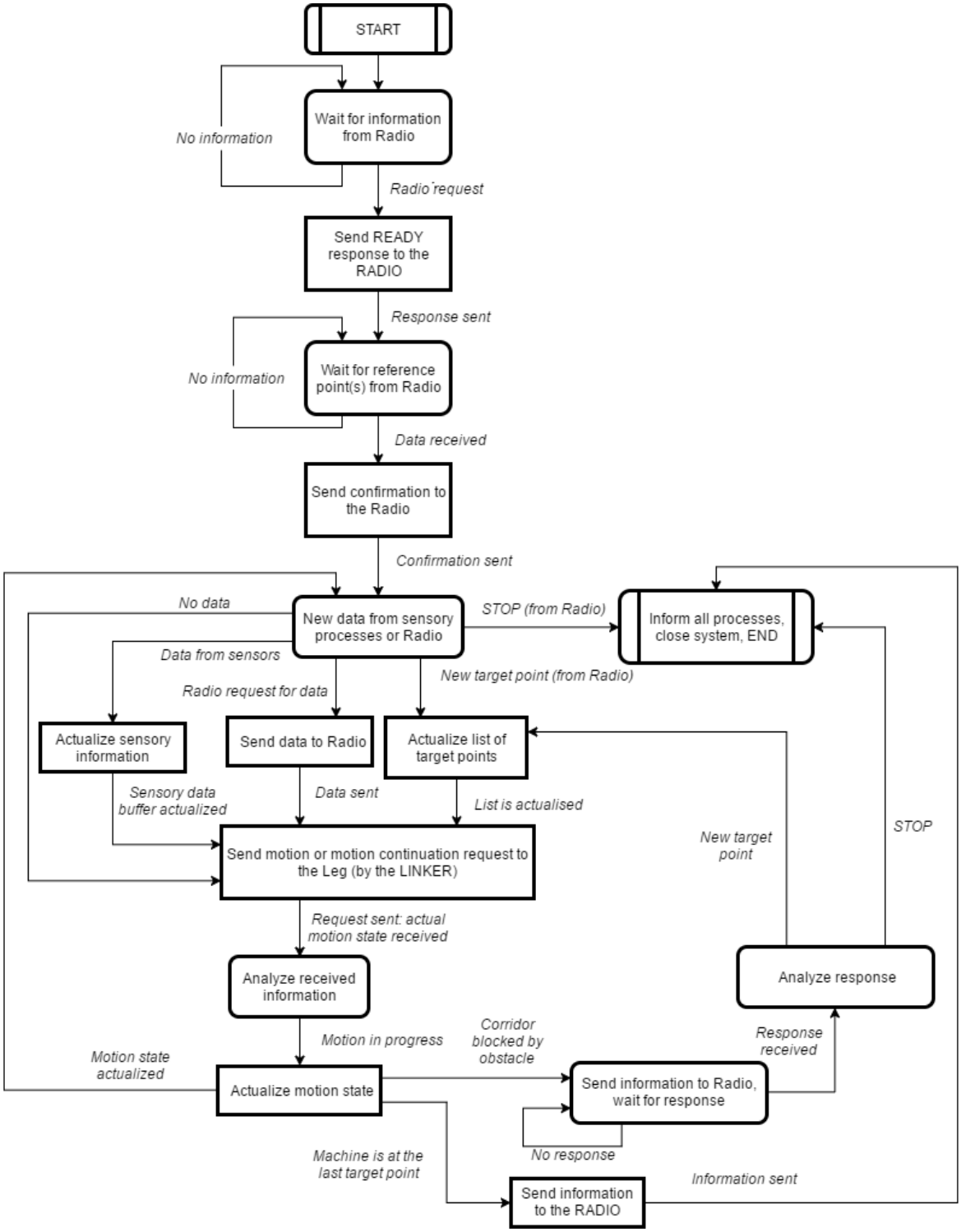


Figure 3-12: Process flow for the brain process of the GROVEN hexapod [15]

can be seen in Fig. 3-5. It divides the activities of the subsystem into different actions which are represented as FSM states. Instead of following a sequential check on whether the demand has been met, it transitions between states based on logical expressions. There is no need to specifically check all specified conditions. The transitions are fired automatically when a certain condition is met. This reduces the load on the processor.

The GROVEN hexapod makes use of a network of processors to implement the processes. They all need to synchronize with each other for adequate data communication. This requires a strict synchronization method to be specified. However, in the proposed system, a central data communication structure called the whiteboard is used (see Sec. 2.3 for more details).

It must be noted that in FSM approach there is some indeterministic factor, because the order of accessing the data repository depends on how fast the processes are cycling; the data access order is variable. In the QNX based system used for GROVEN machines, the order of taking the data is strictly defined.

Chapter 4

Experimental Setup and FSM Implementation

This chapter describes various experiments that were performed to verify the functioning of the proposed control system. The detailed description on how to implement the proposed FSMs is also defined.

In order to verify that the proposed control system is sound and can be implemented on an actual walking machine, it needs to be rigorously tested. For this purpose, the logic labelled finite state automaton based control system was implemented on a simulator. The hexapod walking machine was chosen due to its static stability and flexible locomotion. The sensors described in Sec. 3.2 were attached to the robot and the control system was made to navigate in different environments. A more detailed description is given below:

4.1 The Simulator

V-REP (Virtual Robot Experimentation Platform) [31] was used to simulate the behaviour of the control system. V-REP is a cross-platform simulator allowing the creation of scalable, portable and easy to maintain models and scenes. Four different physics simulators (Bullet, Newton, ODE and Vortex) allows the simulation of soft and rigid body dynamics along with collision detection. Diverse selection of powerful and realistic sensors such as proximity and vision sensors

is also available within the simulator. The ability to run scripts using a remote API is beneficial as it allows a convenient mechanism to control the simulation using the MiEditLLFSM tool.

4.2 Hexapod Walking Machine

A hexapod, inspired by the hexapoda sub-phylum of the anthropods, is a six-legged walking machine. Since, only three legs need to be touching the ground to ensure stability, the other legs of the hexapod can be utilized in reaching new foot placements, offering a lot of flexibility in locomotion.

Typically, a hexapod's body shape [32] can be of two types, rectangular and hexagonal. For ease of control, the rectangular body shape (see Fig. 4-1) of the hexapod is used.

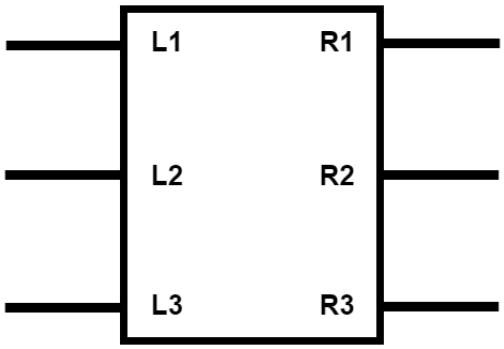


Figure 4-1: The rectangular body architecture for hexapods

There have been many designs for the leg structure [8, 33–35]. The most widely used design is a 3 degree of freedom (DOF) leg (see Fig. 4-2 for the kinematic structure of the leg). This structure gives a total DOF of 18 to the hexapod. Servo motors are a popular choice for the actuation of the leg joints. Thus, a similar leg structure was used for the simulation.

Fig. 4-3 shows the hexapod robot that was used to simulate the behaviour of the control system.

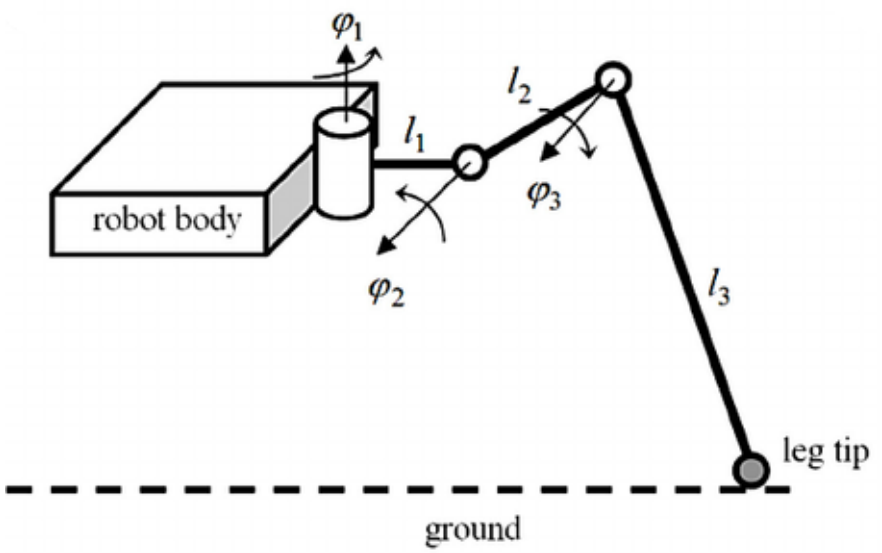


Figure 4-2: The leg structure of the hexapod

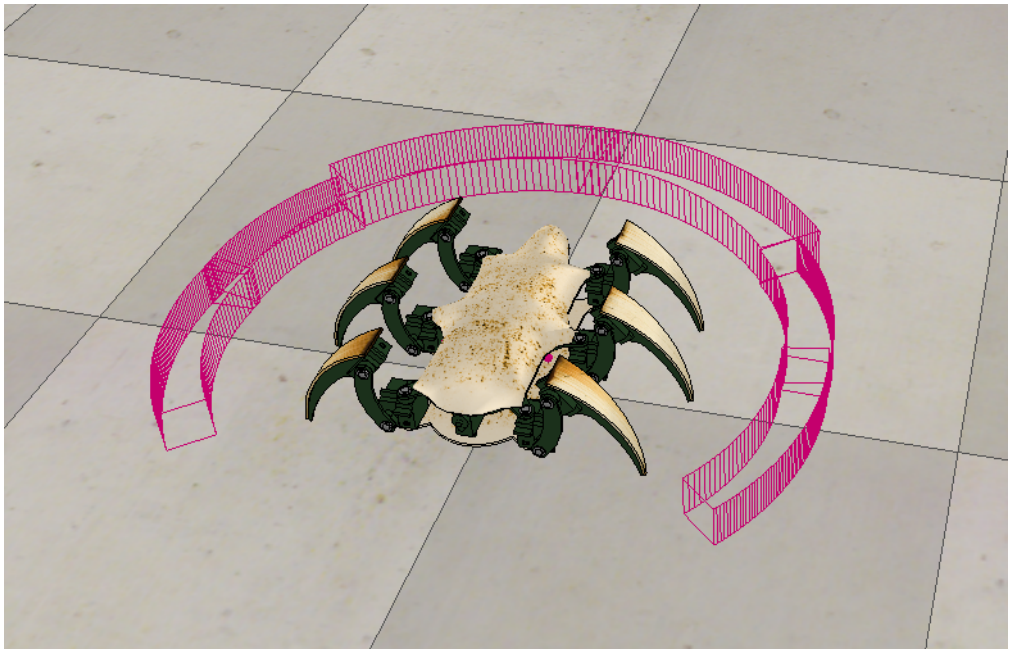


Figure 4-3: Simulation of the hexapod robot

4.2.1 Gait Motion

The hexapod's locomotion is characterized by a series of leg displacements known as a gait. There are many possible gaits for a hexapod. However, for the purpose of experimentation only two periodic gaits were incorporated in the control system.

- **Tripod Gait:** In this gait, three legs stay on the ground while the other three move forward. In a single gait cycle, the hexapod moves two steps. This is the fastest gait and works well with flat surfaces.
- **Wave Gait:** In this gait, only one leg moves at a time. The support polygon is largest among all hexapod gaits making it the most stable gait. However, due to a larger gait period, it is also the slowest gait cycle. Its stability makes it suitable for uneven terrains.

The leg sequence of the tripod and wave gait can be visualized using their gait diagrams shown in Fig. 4-4 and 4-5 respectively. The shaded region in the gait diagrams indicates the swing phase where the leg is in motion while the unshaded area indicates the stance phase i.e. the leg is on the ground.

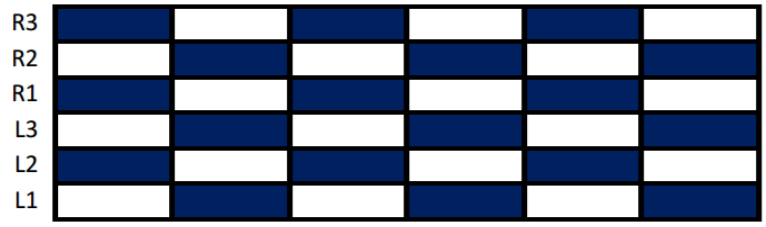


Figure 4-4: Gait diagram for tripod gait

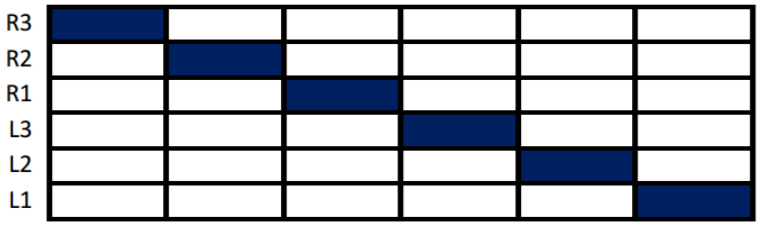


Figure 4-5: Gait diagram for wave gait

4.3 Proximity Sensors

There are several types of proximity sensors available in the V-REP simulator. However, for the purpose of this research, a disc type ultrasonic sensor was selected. The schematic of the sensor is shown in Fig. 4-6.

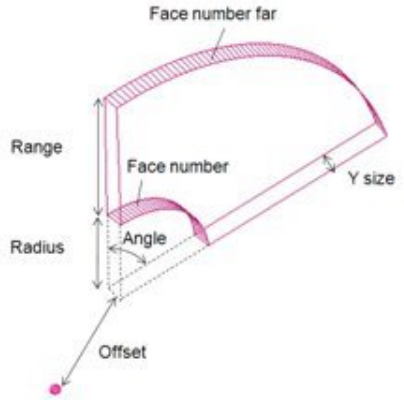


Figure 4-6: Proximity sensor schematic [31]

The hexapod is equipped with six proximity sensors; two at the front, two on the left and two on the right as shown in Fig. 4-7. The angle of the sensors was selected to be 50° and they were arranged in such a way that the field of view of adjacent proximity sensors was overlapping. This was done to ensure the whole view is covered. The range of the front sensors was set as 7.5cm with a radius of 30cm while the side sensors had the same radius but a smaller range of 5cm. This means that the front and side sensors can detect an object up to 37.5cm and 35cm away from the robot centre respectively. The front sensor range was kept larger than the side sensors based on observations during the experimentation process.

4.4 Navigation Conditions

Fig. 4-8 illustrates the possible situations for the hexapod navigation. The green arrow shows the heading direction. The status of the paired proximity sensors are collectively mentioned where a $'0'$ indicates no obstacle and a $'1'$ indicates the presence of an object in the robot neighbourhood.

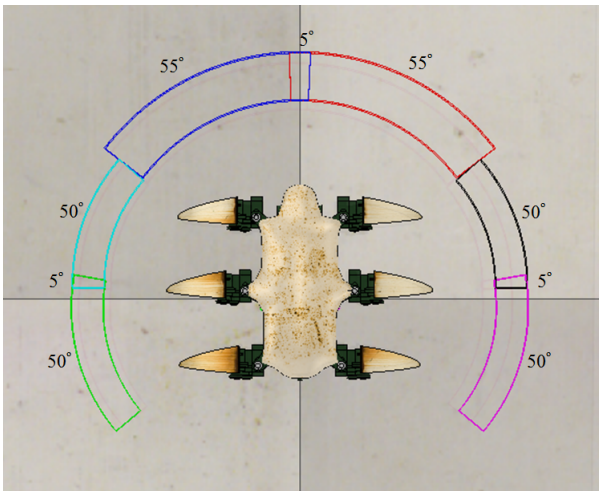
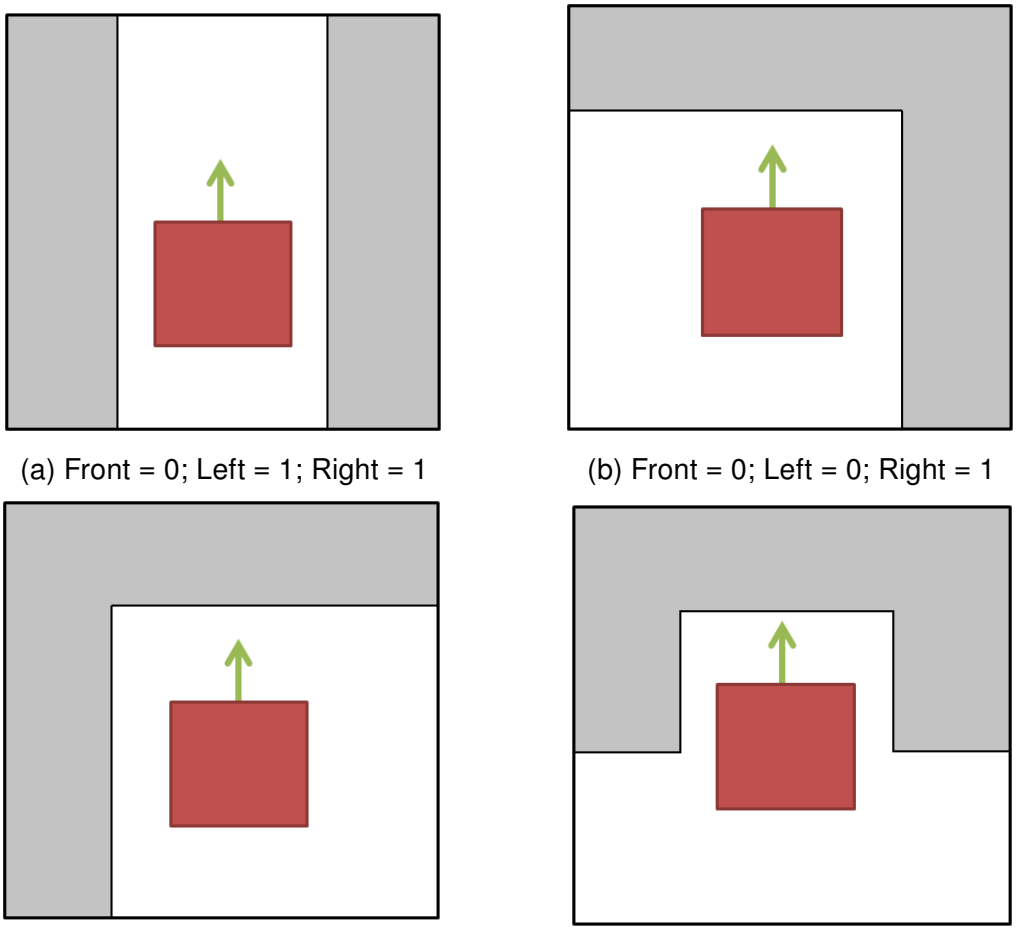


Figure 4-7: Arrangement of the proximity sensors. Red and blue coloured sensors are the front right and left sensors respectively. Black and purple coloured sensors are the right front and back sensors respectively. Cyan and green coloured sensors are the left front and back sensors respectively.



(a) Front = 0; Left = 1; Right = 1

(b) Front = 0; Left = 0; Right = 1

(c) Front = 1; Left = 1; Right = 0

(d) Front = 1; Left = 1; Right = 1

Figure 4-8: Possible scenarios for the hexapod motion

The hexapod control system needs to decide what action to take in all possible situations. The navigation decisions for the situations mentioned in Fig. 4-8 are as follows:

- **Scenario a:** The path in robot heading direction is free so the robot should walk forward.
- **Scenario b:** The path in robot heading direction is blocked by an obstacle. So, the robot should walk in the left direction until the front path becomes free again.
- **Scenario c:** The path in the forward and left direction is blocked. So, the robot should walk towards the right until the forward path becomes free or an obstacle is detected in the right proximity.
- **Scenario d:** The robot is completely trapped so the control system should notify the remote operator that the path is locked and await for further instructions.

4.5 Interface

A Qt based user interface was developed within the V-REP simulation environment as shown in Fig. 4-9. It follows the specifications mentioned in Sec. 3.3. The interface can be displayed together with the simulator view as illustrated in Fig. 4-10.

The simulator environment provides a mechanism to visualize the robot behaviour. The default simulator view shows the top projection of the simulation environment covering an area of 10 x 15 m. The view is adjustable and can be zoomed out up to an area of 80 x 50 m.

The operator can specify target positions for the robot as x and y coordinates measured in meters. These waypoints need to be specified as comma separated coordinates. An example of the mission message is "5,0,7,2,11,-9" where (5,0), (7,2) and (11,-9) are the target positions. There is a control button (*Send Waypoints*) to send the waypoints to the robotic control system.

Similarly, using a control button (*Get Location*), the user can request the

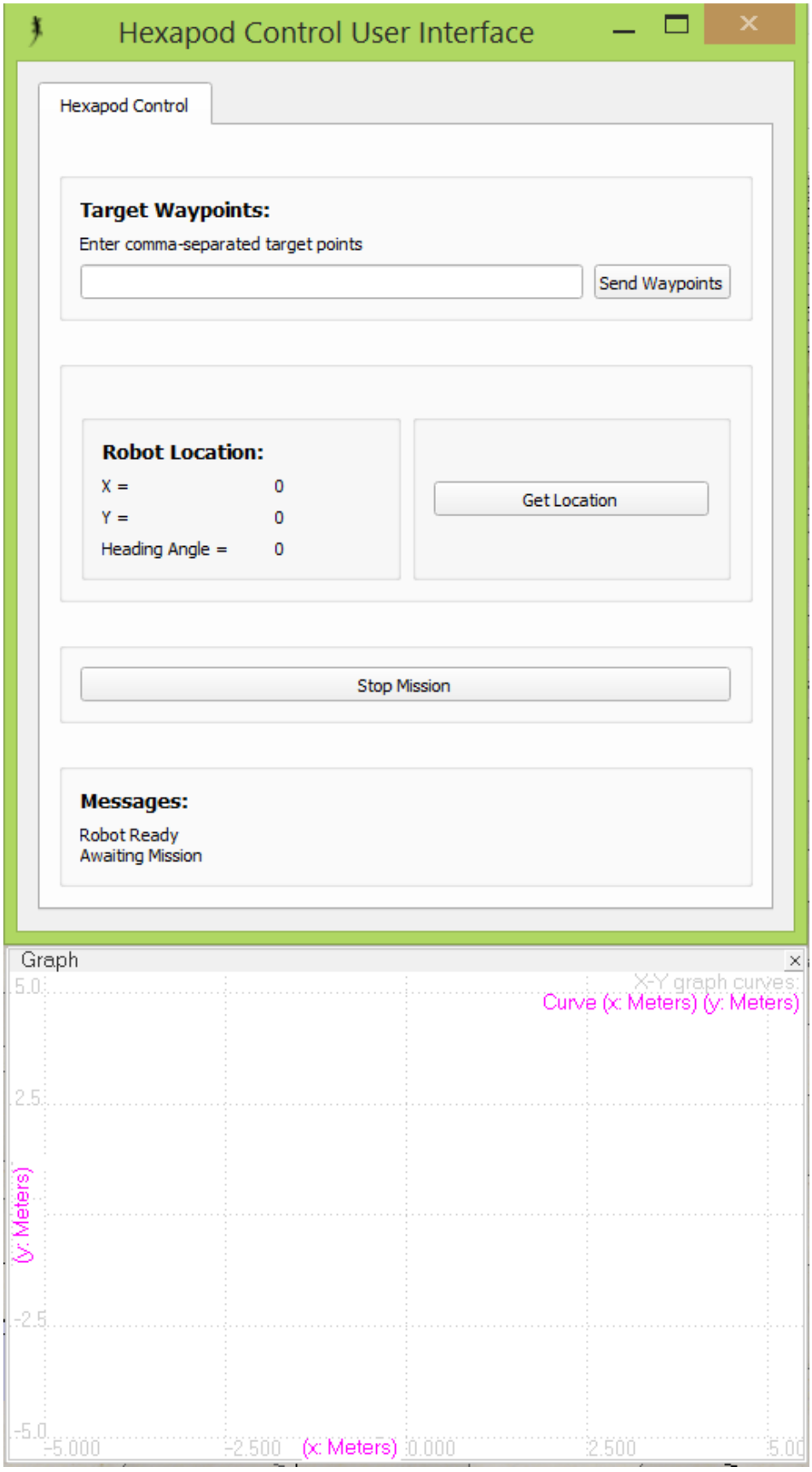


Figure 4-9: The developed user interface

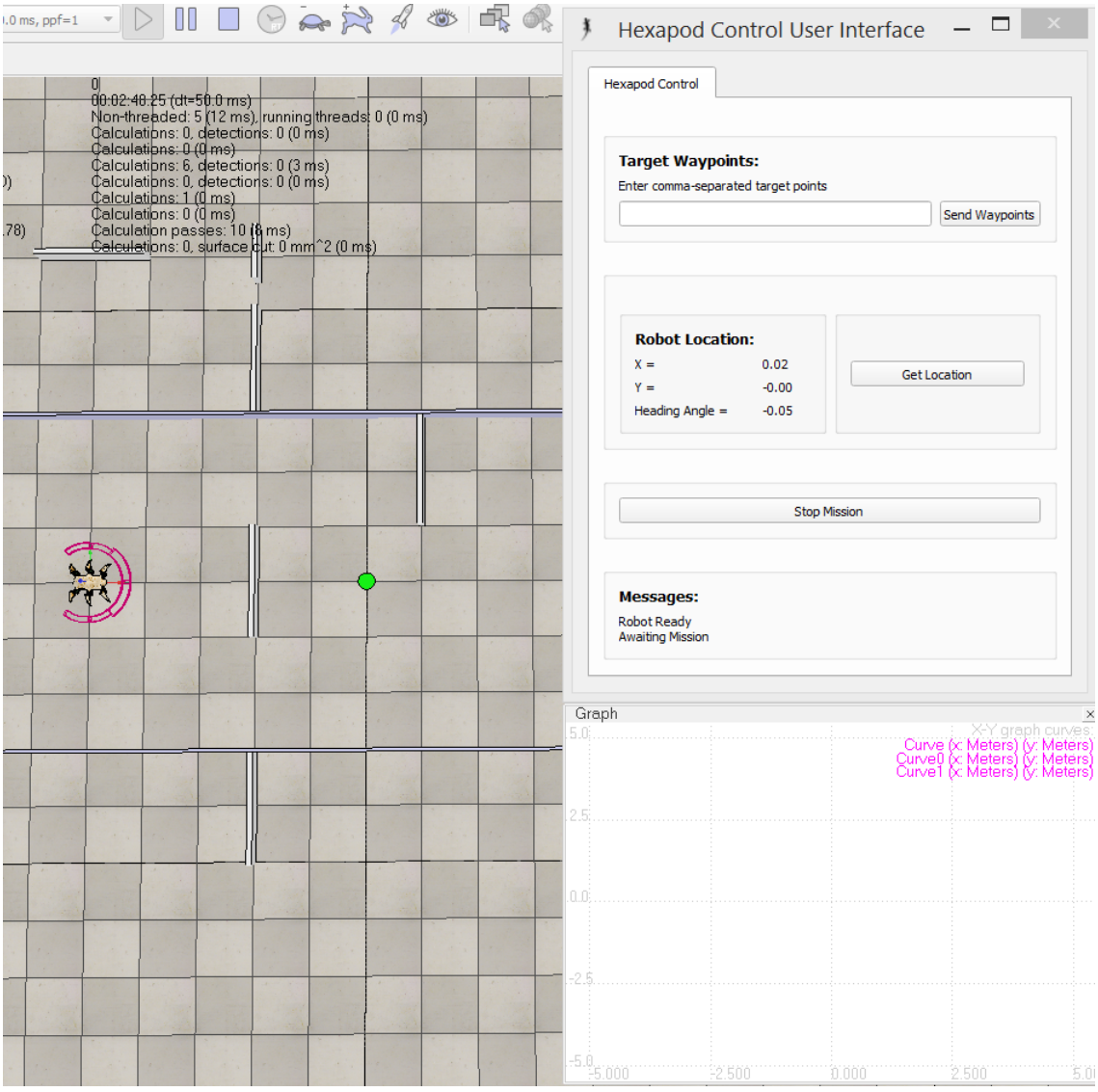


Figure 4-10: User interface along with the simulator view

current position of the robot at any point in time. The x and y coordinate along with the robot heading direction are updated on the graphical user interface when the request is made.

The robot trajectory is also generated for the operator to visualize the robot's movements. The buffer size was set so that the plot shows the robot's trajectory for the past 15 minutes of activity. It should be noted that the hexapod's size is approximately 30 x 35 cm (including the legs), but the generated trajectory shows the hexapod as a point object.

A *Stop Mission* button was included so that the operator may abort the mission if needed. Path locked and fault information is also communicated to the operator via pop-up messages.

Various messages that require a response from the user are displayed as pop-up messages. Fig. 4-11, 4-12 and 4-13 shows the control system messages for a mission completion, a locked path and a system failure respectively.

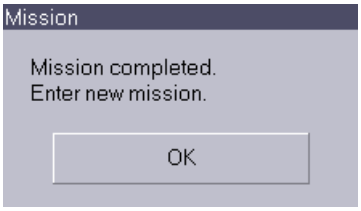


Figure 4-11: Pop-up message for mission completion

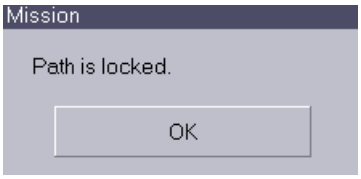


Figure 4-12: Pop-up message for a locked path

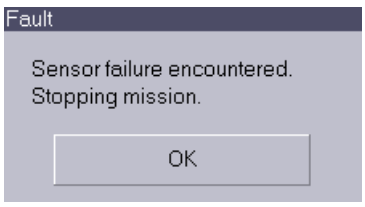


Figure 4-13: Pop-up message for a system fault

4.6 Environments

The environment used to simulate the robot behaviour is composed of an open space. The robot will navigate through it, trying to reach different target positions as specified by the remote operator. A typical environment is shown in Fig. 4-14.

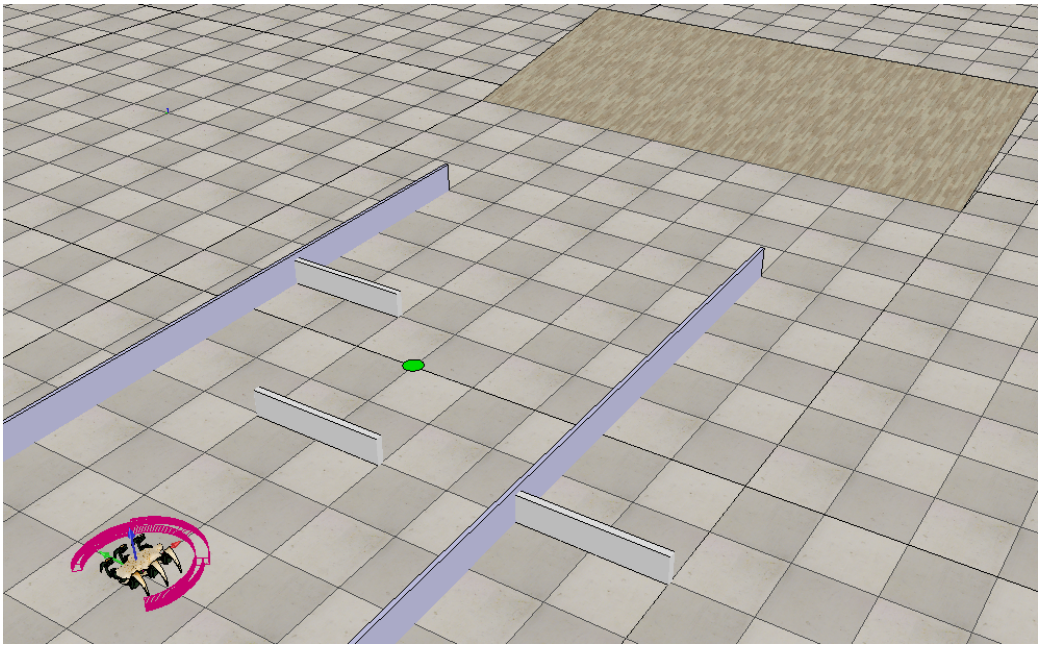


Figure 4-14: Typical environment. The individual size of the visualized squares representing the walking surface is 0.5 x 0.5 m

Objects of variable dimensions and shapes were placed in the robot path to test the obstacle avoidance algorithm. Also, different surface slopes were used to check the gait selection functionality.

The green circle seen in Fig. 4-14 is an indicator for the current goal position while the big wall structures are the corridor limits. They are not physical objects and cannot be detected by the proximity sensors. They are only included to provide a reference to the current target position and corridor limits. Both of these objects move with each new target position. A width of 3m is used for the corridor.

4.7 Implementation of the FSMs

The control structure described in Sec. 3.1 was used as a basis for defining the individual subsystems required for the simulation. Fig 3-2 presents the adapted structure of the hexapod control system. It is composed of three sensory subsystems (*Read Position, Read Orientation, Read Proximity*) and the three hierarchical subsystems (*Global Navigation, Local Navigation, Actuation Control*).

The *Read Position* subsystem acts like a GPS process. It is responsible for tracking the position of the robot in the world frame. The *Read Orientation* subsystem performs data acquisition of the orientation sensors (inclinometers and compass). The last sensory subsystem, *Read Proximity* is responsible for assisting the navigation algorithm by detecting the presence of objects in the robot environment.

The *Global Navigation* subsystem communicates with the remote operator via the user interface subsystem (see Sec. 4.5 for more details) and sends the user demands to the next layer in the hierarchy. The Local Navigation subsystem is mainly responsible for carrying out the limited knowledge trajectory planning and the position evaluation. The lowest level in the hierarchy, *Actuation Control* prepares the motor commands. It should be noted that the inverse kinematics was not explicitly calculated. Instead, the hexapod legs were set to *IK mode* in the simulator and only leg-end trajectories were specified.

The above mentioned FSMs were implemented using the MiEditLLFSM tool. The actions of the individual states were defined using C++ code. When all the activities of the individual FSMs were specified, the executable C++ code was automatically generated by the tool. The execution of the FSMs were determined by a scheduler. Concurrency is maintained by the scheduler by ensuring only one step of the FSMs is executed in each time slot. More details for the real-time scheduling and execution are provided in Sec. 4.8.

During the implementation, all three sections of the state were used (see Sec. 2.2 for details). The *OnEntry* section associated with each state was used to setup the state variables. The *OnExit* section was used for the clean-up code.

All the cyclic actions were implemented in the *Internal* section.

4.8 Real-time Implementation

It is vital for the developed control system to be able to respond to external stimuli within a time constraint. When an obstacle is sighted by the proximity sensors, the robot should be able to alter its trajectory immediately. A delayed response may result in collision and possible damage to the robot structure.

All FSMs described previously are executed simultaneously using a parallel programming structure. Each FSM is executed as a single threaded process. The concurrent FSMs are arranged in a round robin structure following the guidelines provided in [22]. This implies that each FSM gets the processor's resources for a ringlet (a pass over the cycle) execution. Once a ringlet completes its execution, the next FSM in the arrangement gets the processor's resources. This execution is governed by an execution token. When a FSM enters a suspended state, the execution token is passed to the next FSM in the arrangement.

During a ringlet, a local copy of the variables is made before execution of any section of the state. This ensures that the data within a FSM is current and that no other concurrent FSM modifies the data during the state action execution, barring the need for semaphores and mutexes. This subsequently, avoids thread starvation.

Chapter 5

Experimental Results

This chapter presents some of the results obtained during the experimentation process. Initial testing was done on environments imitating an open space. Single waypoints were sent to the robot via the user interface and the robot behaviour was observed. Fig. 5-1 shows the trajectory the robot took when it was instructed to navigate in an open space. It can be seen that the robot starts moving towards the goal. However, due to initial errors in the heading direction, after sometime, it is no longer moving towards the goal position. So it performs a heading correction to return towards the goal.

As a next step, obstacles of variable dimensions were placed in the robot path in order to verify the obstacle avoidance feature. Fig. 5-2 shows the robot behaviour in this scenario. The shaded areas illustrates the combined field of view of the front proximity sensors. When the robot is in close proximity to the obstacle (the sensor field of view overlaps with the obstacle), it deters from its desired path. It starts walking towards the left until the obstacle is no longer in sight of the proximity sensors. Afterwards, it performs a heading correction by turning towards the goal position and then continuing its forward motion. However again the obstacle comes inside the field of view of the front proximity sensors. So, it again exhibits a leftward motion. This continues until there is no obstacle in its trajectory. A similar behaviour was observed for a centrally place obstacle as shown in Fig. 5-3. For perceptual spontaneity, the sensor field of views are not shown in the subsequent robot trajectories.

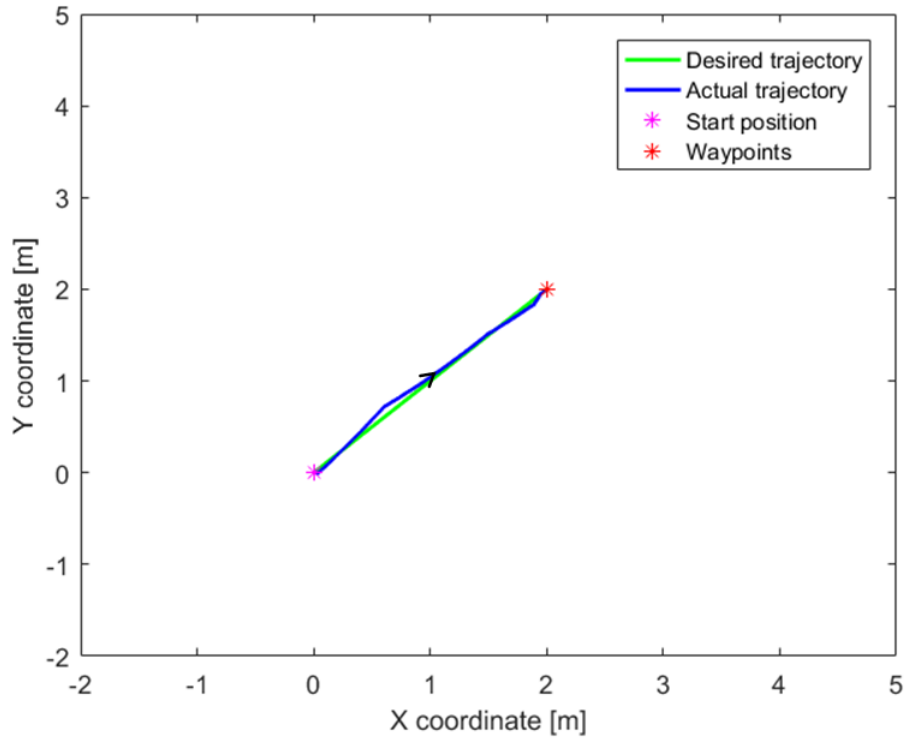


Figure 5-1: Robot trajectory for an open path

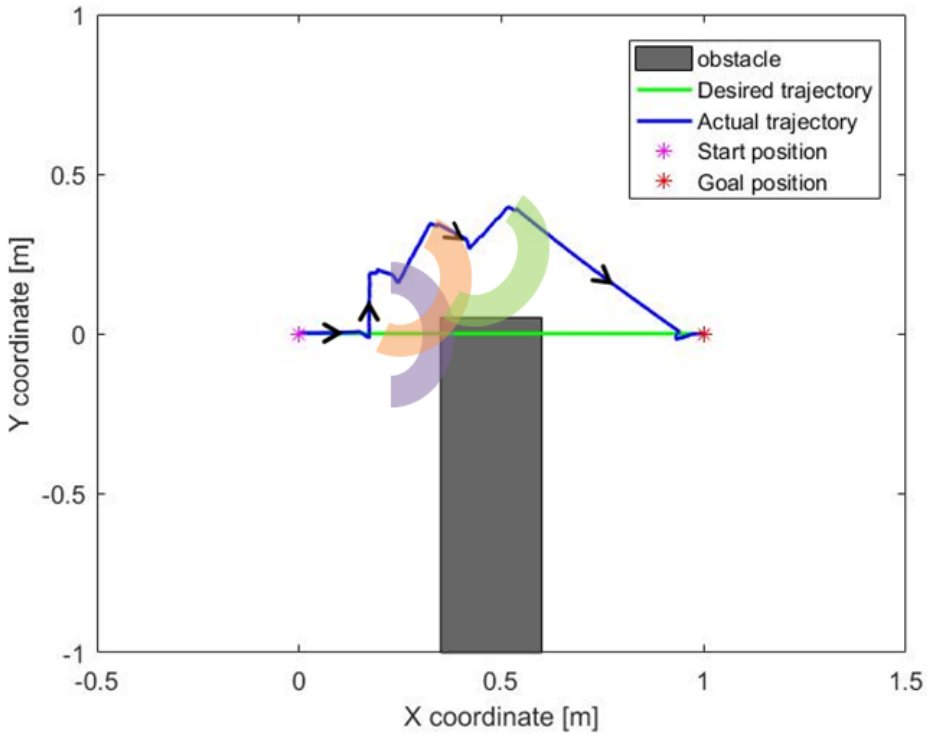


Figure 5-2: Robot trajectory for a path with an obstacle

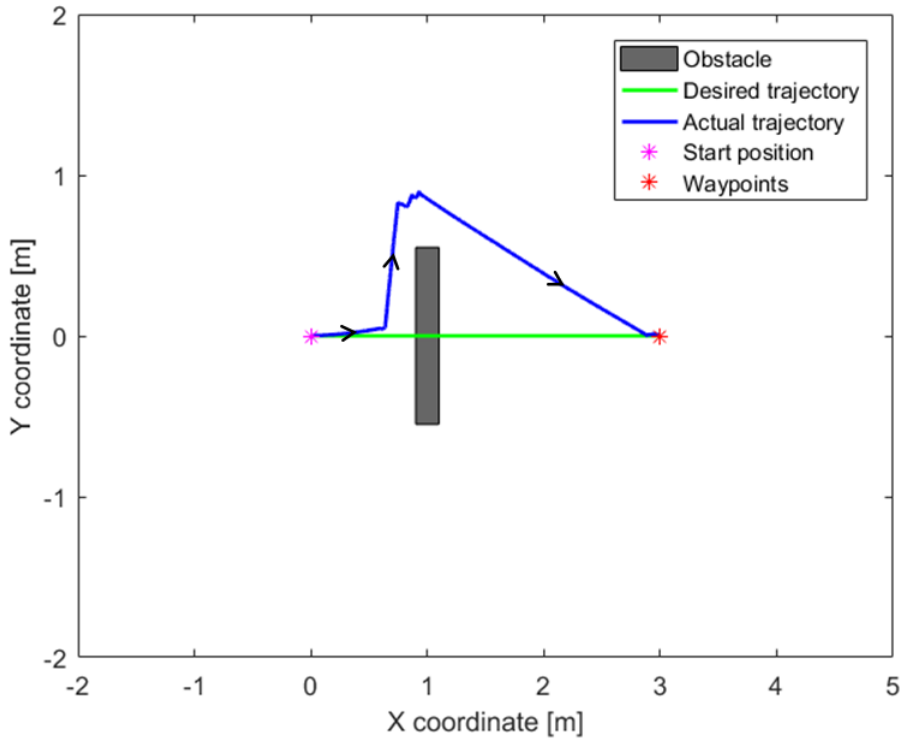


Figure 5-3: Robot trajectory for a path with an obstacle placed centrally

In another tested scenario, obstacles were placed at an angle to the desired trajectory. This was done to see how the control system behaves since only the robot can move forward, left or right and not specifically at an angle. However, the control system was able to navigate around the object using a zig-zag motion (i.e. one step forward followed by one sideways step). The robot trajectory for this scenario is shown in Fig. 5-4.

The next step was to test whether the walking machine can switch between the tripod and wave gait. An inclined surface was placed in the robot path. Initially, the robot moves with the tripod gait. Once the inclinometers indicate the presence of an inclined surface, the robot shifts to the wave gait which is more stable and prevents the robot from toppling over. The behaviour of the robot in this scenario is shown in Fig. 5-5.

The last step in the initial testing was to include the corridor width in the robot motion.The path around the obstacle was blocked from the left side. In this case, the robot first tried to walk leftwards but detected no path existed in that direction as it had reached the corridor limit so it started to move towards the right until a

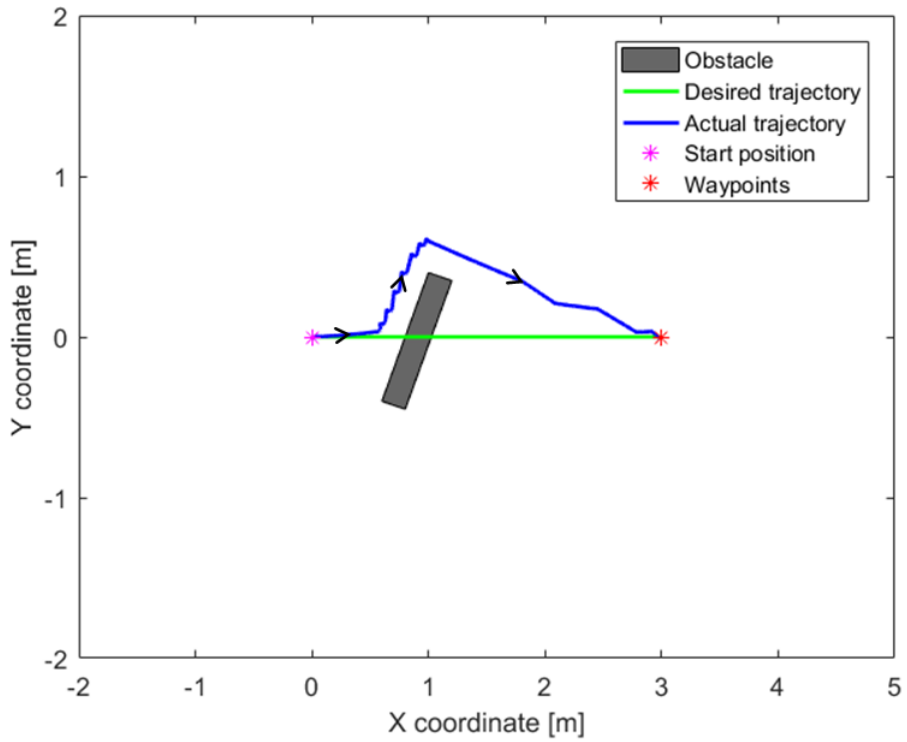


Figure 5-4: Robot trajectory for a path with a slanted obstacle

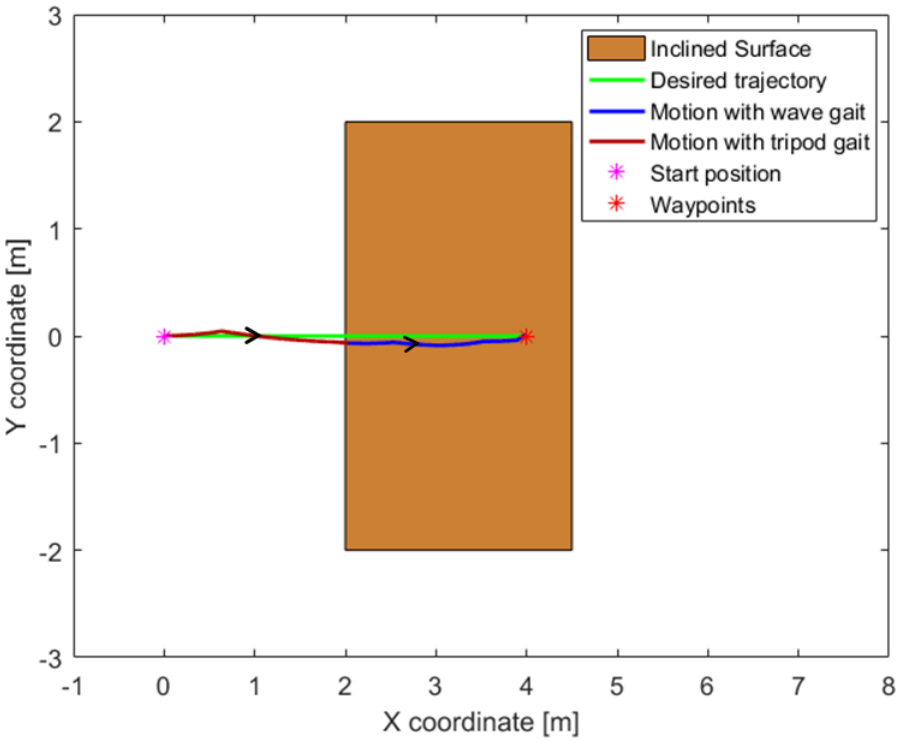


Figure 5-5: Robot trajectory for a path with an inclined surface

free path was found. Fig. 5-6 demonstrate this behaviour.

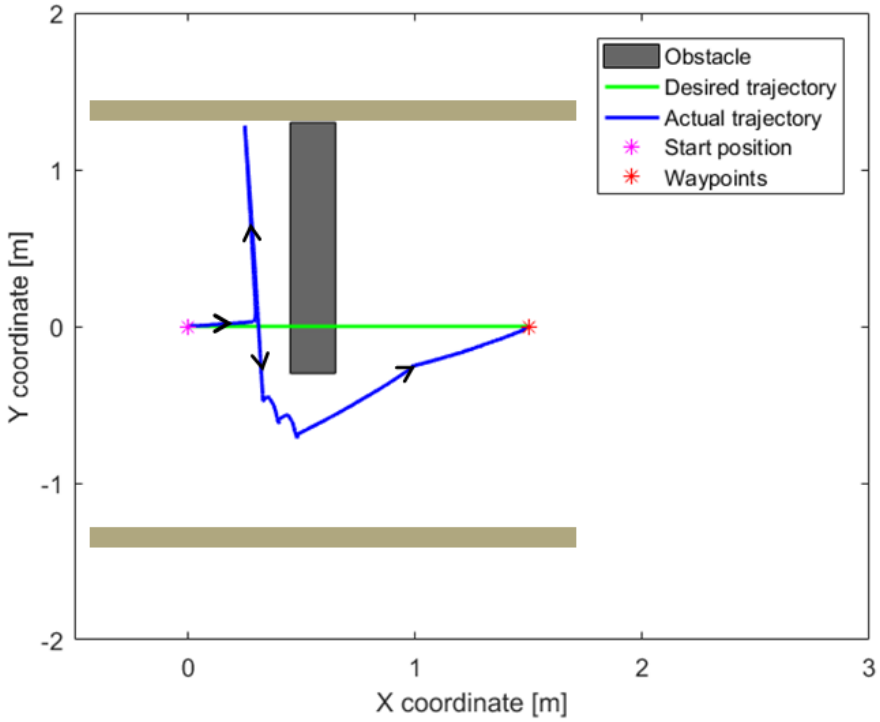


Figure 5-6: Robot trajectory for a path with an obstacle blocking leftward motion

A situation could arise where the path to the goal position does not exist within the corridor limits. A large obstacle was placed in the robot's path to see how it behaves. Fig 5-7 illustrates this scenario. The robot first moves to the left when it detects the presence of an obstacle. When it fails to find a free path towards the left, it moves towards the right. However, no path exists from the right side either. So, the robot stops and informs the operator that the path is locked and asks for further instruction. It can be observed from the map that there is a free path towards the goal. This is in reality not true as a physical robot is not a point object and its dimensions need to be included in the realization.

Once, the initial testing showed promising results, advanced scenarios were tested. Multiple waypoints were sent to the control system. Such a scenario is depicted in Fig. 5-8. The robot passes through all the specified waypoints. The corridor limits are only shown in the subsequent robot trajectories when they play a part in the robot motion.

Next, multiple objects were placed in the robot's path as depicted by Fig.

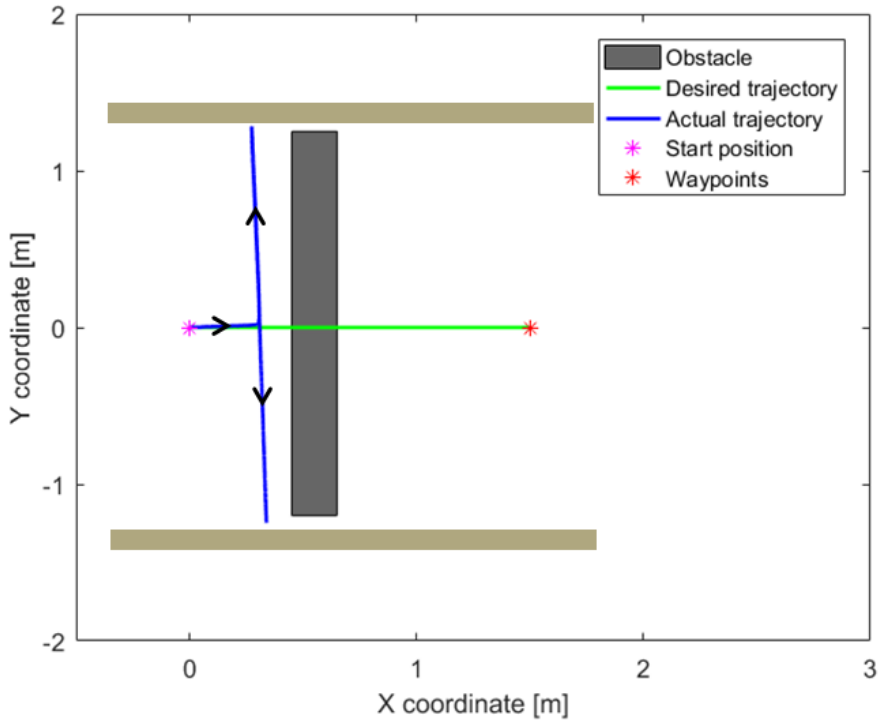


Figure 5-7: Robot trajectory for a locked path

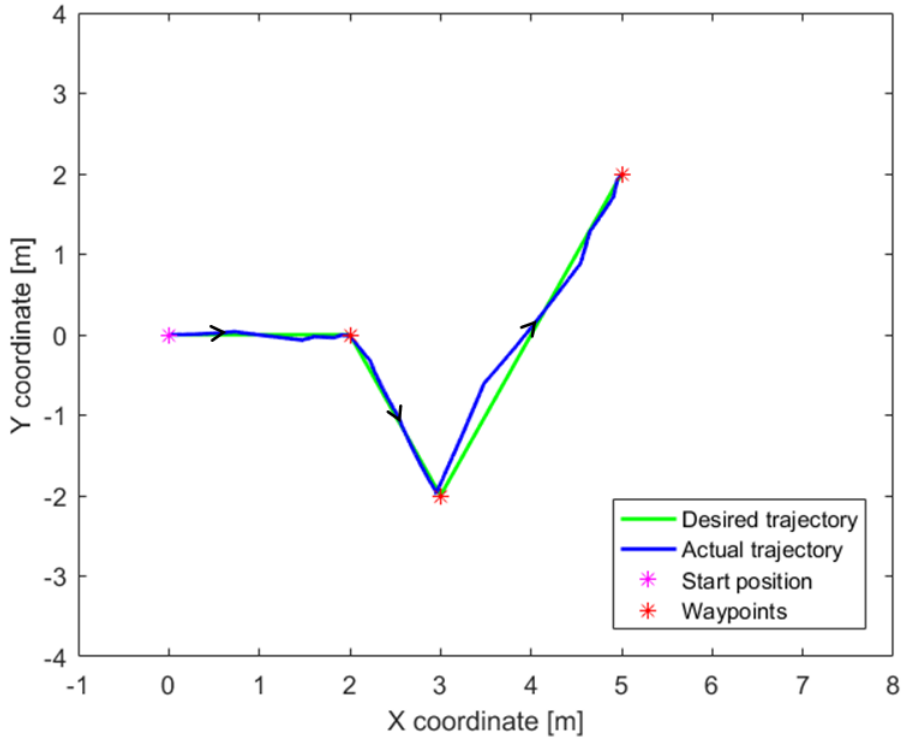


Figure 5-8: Robot trajectory for an open corridor path

5-9. The robot control system manages to navigate around the obstacles while passing through the specified waypoints.

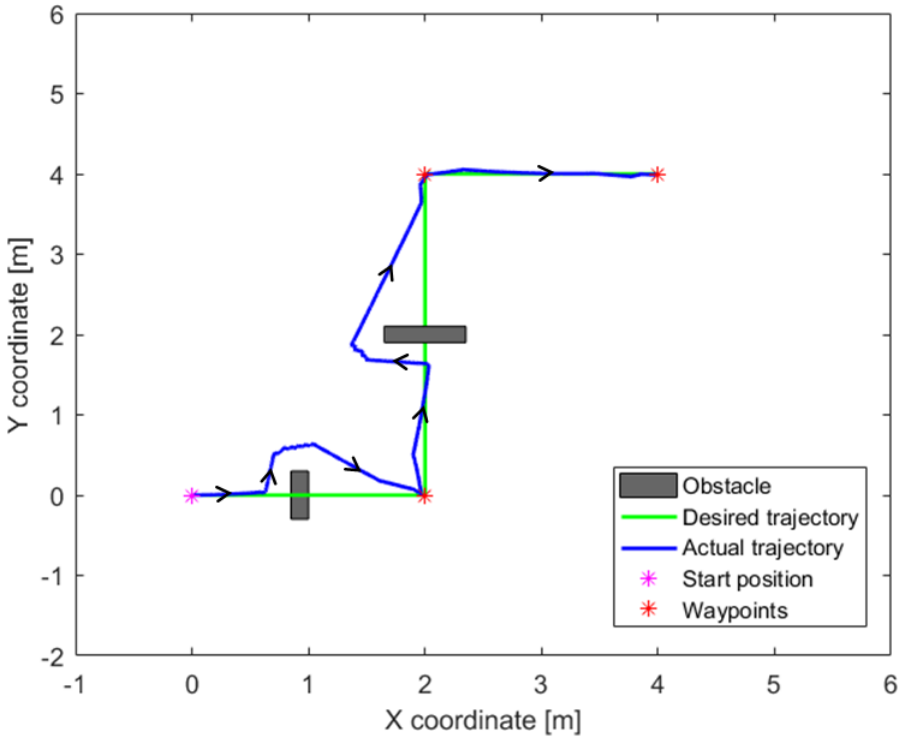


Figure 5-9: Robot trajectory for a corridor path with obstacles

Lastly, the path to one of the waypoints was blocked by a large obstacle. Fig. 5-10 highlights this scenario. When the robot was unable to navigate to the desired target point, it sent this information to the operator. A new waypoint was sent to the control system instead. The robot then continued its mission as specified by the new goal points.

It is possible to modify the mission target points during the robot motion as well. Such a scenario was tested next as demonstrated in Fig. 5-11. Initially, the mission comprised of two waypoints; [4,0] and [4,-2]. However, while the robot was still moving towards the first waypoint, the second waypoint was modified and sent to the control system. The robot was able to accommodate the change and moved to the new waypoint [4,2] instead after reaching the first waypoint.

The last phase of the experimentation was to test whether the system is able to handle system faults. Since the implementation of the control system is done on a simulator, the fault scenarios were artificially created. The user interface

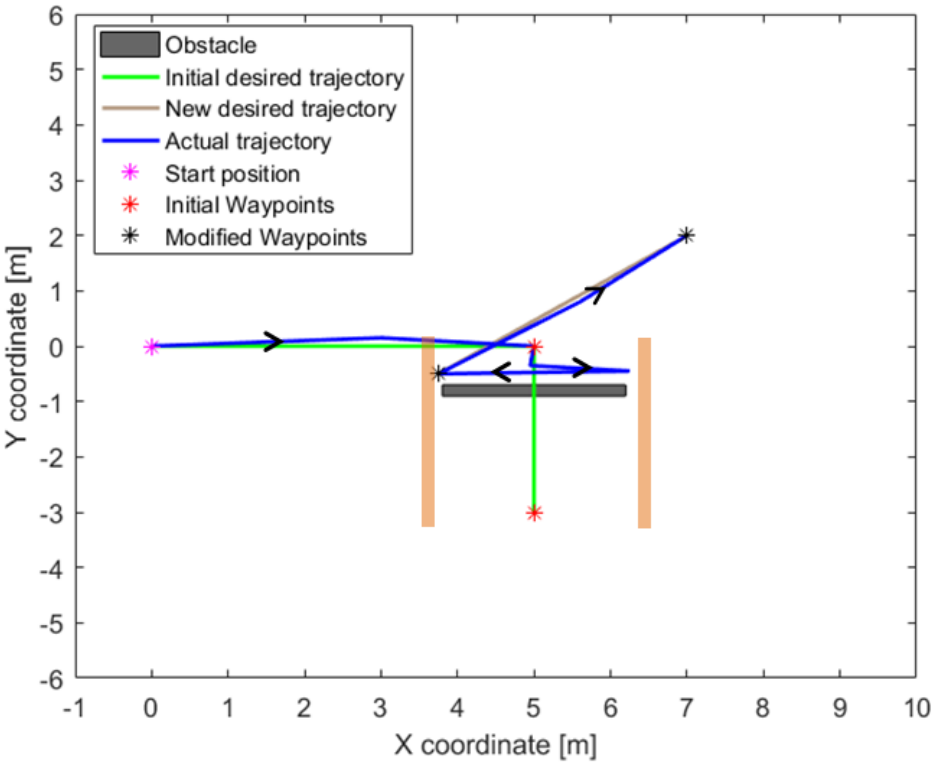


Figure 5-10: Robot trajectory for a blocked corridor path

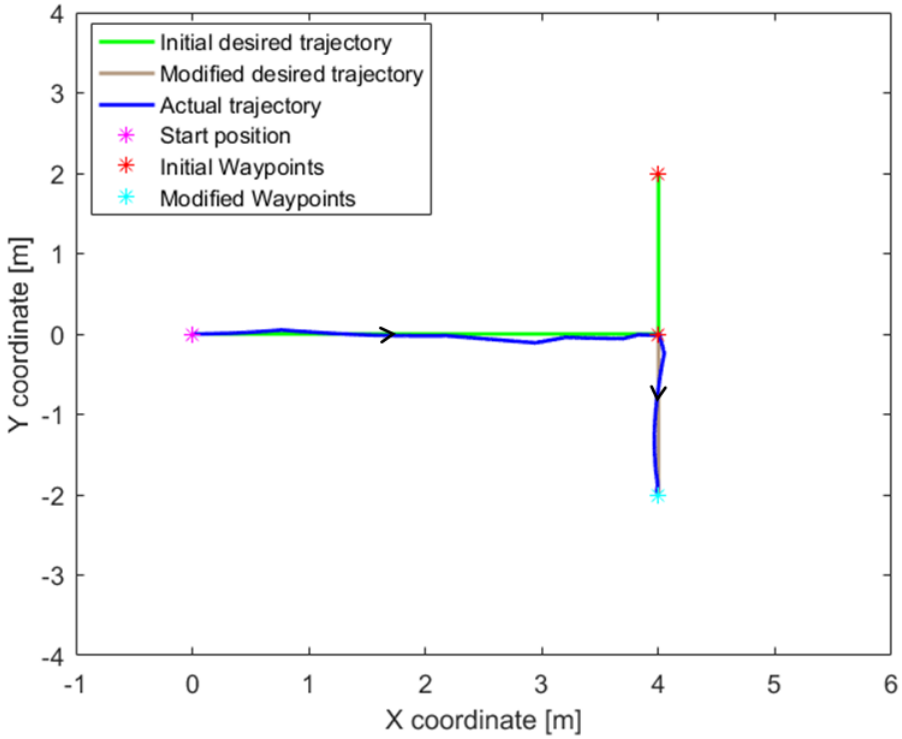


Figure 5-11: Robot trajectory for an online modified mission

was modified and a new section was included (shown in Fig. 5-12) to produce the system faults at desired points in time.

Figure 5-12: Modified user interface for the system faults

The robot was provided with a mission and at random instances, system faults were fabricated. One such instance is demonstrated in Fig. 5-13. When the fault was produced, the robot stopped its mission and informed the user that it had encountered a system failure.

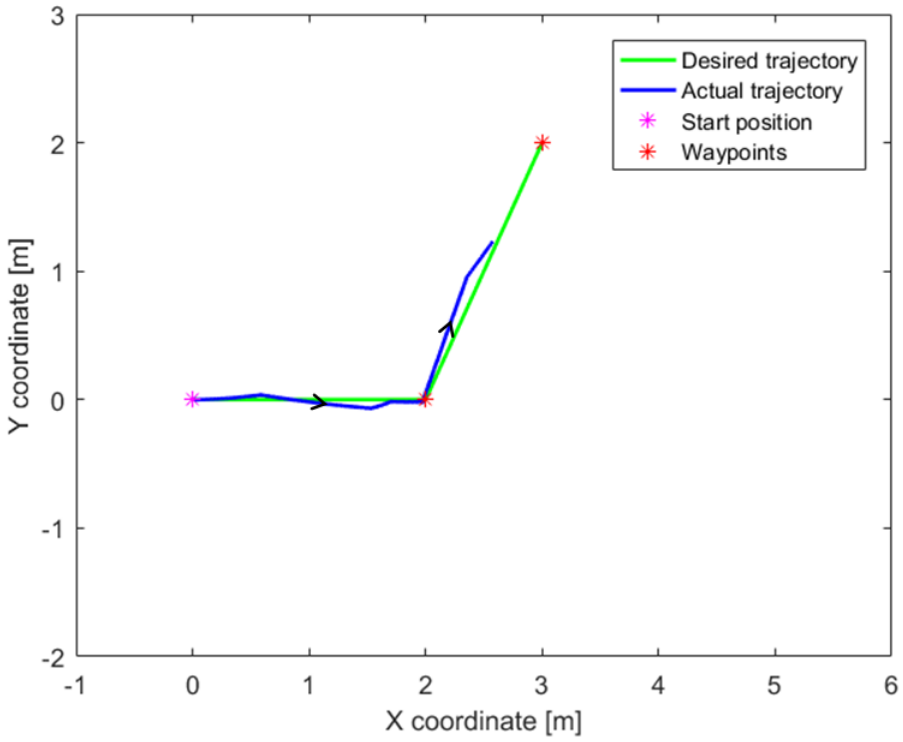


Figure 5-13: Robot trajectory for a mission suffering from a system fault

5.1 Discussion

The FSM based control system was tested for a hexapod walking machine navigating in an unknown environment. The waypoints were specified by a remote operator and the robot behaviour was observed. The control system was able to perform a limited knowledge path search and navigated towards the user-defined waypoints.

In ideal cases, a smooth trajectory is usually preferred. In contrast, the robot was seen to exhibit a slightly erratic trajectory. This is due to the fact that a small tolerance of $\pm 5^{\circ}$ has been included in the robot heading direction error. Increasing the tolerance results in a smoother robot trajectory. However, increasing the tolerance too much causes a more erratic behaviour near the goal position. Thus, a compromise needs to be made.

A position error of $\pm 5cm$ was observed for each waypoint during the simulation experiments. This error is due to the fact that the hexapod can only move at discrete distances corresponding to its stride length.

It was observed during the testing phase that the navigation algorithm used by the control system in some scenarios concluded that the path is locked, even though a free path existed. This situation occurred when a U-shaped obstacle was placed in the robot path. This can be avoided by integrating a wall-following algorithm in the obstacle avoidance planner.

Chapter 6

Conclusion

This chapter presents a summary of the work done along with the research conclusions and the problems encountered during the implementation phase.

A control system is responsible for a large number of tasks. Implementing all these within one module is difficult. A functional decomposition of the control structure simplifies the implementation and introduces modularity in the system. The implemented control structure focused on modularity which allows flexibility in the system. The same control structure can be adapted to different walking machine prototypes by just modifying the motor control subsystem.

The hierarchical structure (see Sec. 3.1 for details) allowed distribution of tasks among the subsystems. The high level control subsystem generates a global strategy for the robot locomotion. While the lower layer generates the detailed path in real-time. The FSM based approach was chosen to implement the individual subsystems as it offers a quick to design and implement solution due to its simplicity.

The most critical component in control system development is defining the system specifications. All possible scenarios that the control system may encounter need to be identified and appropriate handles should be incorporated into the control structure. The system specification needs to follow an iterative scheme as it was sometimes observed during the implementation and testing phases that the system specification needed modification to handle certain unforeseen scenarios.

All modules developed in this research have distinct functions but rely on each other for inputs. This interdependency of the modules governs how adequately the control system reacts to external stimuli. The inter-modular communication was achieved through a central data repository. The identification of the data (variables and their data types) that the central repository must hold is also crucial for the success of the control system. The data repositories need to be set-up properly so that all relevant information is accessible to individual modules. Furthermore, the cyclic processes need to produce data at sufficient rates so that the repositories are always up-to-date.

The system specification and data repository definition was identified as the most difficult part in the control system development as initial designs did not focus much on this element resulting in improper behaviour of the control system.

Generally, FSM based approaches use an asynchronous event-driven multi-threaded model. However, such systems require a supplementary process to manage the synchronization of the threads which increases the load on the processor. Although such approaches may be ideal for the IT industry, but in real-time robotics applications, the load on the processor should be minimized. Thus, a single threaded FSM execution approach was used to implement the FSMs.

Real-time systems need to ensure the response is produced within a time constraint. A single threaded FSM implementation without preemption may cause thread starvation. The frequencies of each periodic process (i.e. the sensory subsystems) need to be set carefully in order to avoid this. However, during the testing phase, no such scenario was encountered.

Rigorous testing of the control system behaviour is an important step in control system development. All possible scenarios need to be simulated and robot behaviour needs to be analysed. To make sure that the individual finite state machines are interacting with each other appropriately, the active states of all automatons need to be observed at each time slice. Based on these observations along with the robot trajectories, the correctness of the system was determined.

The proposed approach makes use of shared variables/memory. The problem with such an approach is that when one process is reading the shared mem-

ory, another process may alter the contents of that variable. This generally requires the use of mutexes and semaphores to protect the data. In contrast, a single sequential scheduler is used in the MiEditLLFSM tool which ensures that in a single time slice, only one FSM accesses the whiteboard/shared memory. Thus, there is no contention among the FSM processes during their execution.

Typically, when a thread is executing, all the compiled instructions need to be loaded in the processor memory. However, with a FSM based implementation only the executable code of the current state needs to be loaded into the processor memory. Consequently, less processor memory is required by the control system resulting in a low processor overhead.

Overall, the developed control system yielded satisfactory results. The robot was able to respond to the presence of obstacles in its environment almost immediately. No collision was encountered during the testing phase. Thus, the logic labelled finite state automaton approach is a promising technique for developing the control system of a walking machine.

Bibliography

[1] K. Pearson (1976) The control of walking, Scientific American 235(6): pp. 72-86.

[2] R. Brooks (1986). A robust layered control system for a mobile robot, IEEE journal on robotics and automation 2.1, pp. 14-23.

[3] D. Payton (1986). An architecture for reflexive autonomous vehicle control, IEEE International Conference on Robotics and Automation, vol. 3, pp. 1838-1845.

[4] A. H. Cai, T. Fukuda, F. Arai, T. Ueyama and A. Sakai (1995). Hierarchical control architecture for cellular robotic system-simulations and experiments, IEEE International Conference on Robotics and Automation, vol. 1, pp. 1191-1196.

[5] L. Minati, and A. Zorat (2002). A tree architecture with hierarchical data processing on a sensor-rich hexapod robot, Advanced Robotics, 16(7), pp. 595-608, DOI: 10.1163/15685530260390737.

[6] T. Odashima, Z. Luo and S. Hosoe (2002). Hierarchical control structure of a multilegged robot for environmental adaptive locomotion, Artif Life Robotics 6: 44, DOI: 10.1007/BF02481208.

[7] R. A. Brooks (1989). A robot that walks; emergent behaviors from a carefully evolved network, Neural computation 1, no. 2, pp. 253-262.

[8] T. Zielinska and J. Heng (2002). Development of walking machine: mechanical design and control problems, Mechatronics Int. J. Mechatronics, vol. 12, pp. 737-754.

[9] M. Li, X. Yi, Y. Wang, Z. Cai and Y. Zhang (2016). Subsumption model implemented on ROS for mobile robots, IEEE Systems Conference (SysCon), Orlando, FL, pp. 1-6, DOI: 10.1109/SYSCON.2016.7490651.

[10] J. Simpson, C. L. Jacobsen, and M. C. Jadud (2006). Mobile Robot Control - The Subsumption Architecture and occam-pi, Communicating Process Architectures, volume 64 of Concurxrent Systems Engineering, Amsterdam, IOS Press, pp. 225-236.

[11] G. C. Buttazzo (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Real-Time Systems Series 24, DOI: 10.1007/978-1-4614-0676-1.

[12] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa and C. Taliercio (2008). Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application, IEEE Transactions on Nuclear Science, vol. 55, no. 1, pp. 435-439, DOI: 10.1109/TNS.2007.905231.

[13] D. Hildebrand (1992). An Architectural Overview of QNX, USENIX Workshop on Microkernels and Other Kernel Architectures, pp. 113-126

[14] T. Zielinska (2005). Control and navigation aspects of a group of walking robots, Robotica, vol. 24, no. 1, pp. 23-29, DOI: 10.1017/S0263574705001840.

[15] T. Zielinska and J. Heng (2006). Real-time-based control system for a group of autonomous walking robots, Advanced Robotics, vol. 20, no. 5, pp. 543-561, DOI: 10.1163/156855306776985568.

[16] M. Raibert, K. Blankespoor, G. Nelson and R. Playter (2008). BigDog, the Rough-Terrain Quadruped Robot, IFAC Proceedings Volumes, vol. 41, no. 2, pp. 10822-10825, ISSN 1474-6670, DOI: 10.3182/20080706-5-KR-1001.01833.

[17] R. G. Simmons (1994). Structured control for autonomous robots, IEEE Transactions on Robotics and Automation, vol. 10, no. 1, pp. 34-43, DOI: 10.1109/70.285583.

[18] V. Estivill-Castro and R. Hexel (2015). Logic Labelled Finite-State Machines and Control/Status Pull Technology for Model-Driven Engineering of Robotic Behaviours, J.-C. Rault (Ed.), Presented at the ICSSEA '15, Paris, France.

[19] S. J. Mellor and M. Balcer (2002). Executable UML: A foundation for model-driven architecture. Addison-Wesley Publishing Co., Reading, MA.

[20] F. Wagner, R. Schmuki, T. Wagner, and P. Wolstenholme (2006). Modeling Software with Finite State Machines: A Practical Approach, CRC Press, NY.

[21] David Harel and Amnon Naamad (1996). The statemate semantics of statecharts, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 5 no. 4, pp. 293-333, DOI: 10.1145/235321.235322.

[22] V. Estivill-Castro and R. Hexel (2013). Arrangements of Finite-state Machines - Semantics, Simulation and ModelChecking, 1st International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 182-189, DOI: 10.5220/0004317101820189.

[23] V. Estivill-Castro and D. A. Rosenblueth (2011). Model-checking of transition-labeled finite-state machines, Int. Conf. Adv. Softw. Eng. & its Applications, ser. Comm. in Computers and Inf. Sc., T.-H. e. a. Kim, Ed., vol. 257, Jeju Island, Korea, Springer Verlag, pp. 61.

[24] D. Harel and M. Politi (1998). Modeling Reactive Systems with Statecharts: The Statemate Approach, 1st ed. New York, NY, USA: McGraw-Hill, Inc.

[25] J. E. Hopcroft, R. Motwani and J. D. Ullman (2016). Introduction to Automata Theory, Languages, and Computation, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

[26] D. N. Hartford and G. B. Baecher (2004). Risk and uncertainty in dam safety, Thomas Telford.

[27] V. Estivill-Castro, R. Hexel, and D. A. Rosenblueth (2012). Efficient model checking and FMEA analysis with deterministic scheduling of transition labeled finite-state machines, 3rd IEEE World Congress on Software Engineering, Wuhan University of Technology, Wuhan, China.

[28] V. Estivill-Castro and R. Hexel (2015). Simple, not simplistic the middleware of behaviour models, International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Barcelona, Spain, pp. 189-196.

[29] B. Hayes-Roth (1985). A blackboard architecture for control, Artif. Intell., vol. 26, no. 3, pp. 251-321.

[30] J. Ng and T. Braunl (2007). Performance Comparison of Bug Navigation Algorithms, Journal of Intelligent and Robotic Systems, vol. 50, no. 1, pp. 73-84, DOI:10.1007/s10846-007-9157-6.

[31] "The robot simulator v-rep", http://www.coppeliarobotics.com/, accessed: 2017-15-10.

[32] X. Ding, Z. Wang, A. Rovetta and J. M. Zhu (2010). Locomotion Analysis of Hexapod Robot, Climbing and Walking Robots, Behnam Miripour (Ed.), InTech, DOI: 10.5772/8822.

[33] U. Saranli, M. Buehler and D. E. Koditschek (2000). Design, modeling and preliminary control of a compliant hexapod robot, IEEE International Conference on Robotics and Automation, Symposia Proceedings (Cat. No.00CH37065), San Francisco, CA, vol. 3, pp. 2589-2596, DOI: 10.1109/ROBOT.2000.846418.

[34] F. Tedeschi and G. Carbone (2014). Design issues for hexapod walking robots, Robotics, vol. 3, no. 2, pp. 181-206.

[35] T. Zielinska, T. Goh and C. K. Chong (1999). Design of autonomous hexapod, First Workshop on Robot Motion and Control, pp. 65-69.

Appendix

The proposed means of interprocess communication makes use of a central database where all shared data is stored. This central database consists of several repositories as described in the tables below.

Table 1: Sensor Repository

DATA NAME	DATA TYPE	DESCRIPTION
Proximity Sensors	Boolean Array	Data from the proximity sensors. ’0’ and ’1’ indicate absence and presence of obstacles respectively. Default array size is 6.
Position Sensor	Float Array	Positional data from the GPS sensors. Array size is 2 representing the x and y coordinates.
Compass Sensor	Integer	Data from the compass sensor i.e. yaw angle of the robot. Data is stored as degrees.
Inclinometer Sensor	Integer Array	Data from the inclinometers i.e. roll and pitch angles of the robot. Data is stored as degrees.

Table 2: Waypoint Repository

DATA NAME	DATA TYPE	DESCRIPTION
Mission Targets	Float Array	Array containing goal positions within a mission. Defaults size of mission is 5 target positions.

Table 3: Demand Repository

DATA NAME	DATA TYPE	DESCRIPTION
Demand for Position	String	User demanding current robot location.
Demand for Heading	String	User demands for current robot heading.
Demand to cancel mission	String	Cancel mission request from user.
Demand to modify mission	String	User requesting a modification in mission target points.

Table 4: Status Repository

DATA NAME	DATA TYPE	DESCRIPTION
Evaluated Position	Float Array	Evaluated position using incremental localization. Only x and y coordinates are evaluated.
Heading Error	Integer	Calculated heading error.
Fault Status	Boolean Array	Status of faults that may occur during a mission. Three types of fault are considered. ’0’ indicates no fault while’1’ indicates an occurrence of a fault.
Within Corridor	Boolean	Robot is moving within the corridor limits.
Target Reached	Boolean	Current target position has been reached.
Gait Type	Integer	Type of gait currently used by robot. ’1’ and ’2’ indicates tripod and wave gaits respectively.

To define these data repositories, messages need to be defined in the "gusimplewhiteboardtypelist.tsl" file of the whiteboard directory as shown below. There is a limitation of the whiteboard which requires each message to have a maximum size of 128 bytes. This needs to be taken into account when defining the whiteboard messages.

```
class:SensorProximity,      nonatomic, ProximityOutput,      "ProximityOutput", Class that returns
                                                             obstacle detection from the proximity sensors
class:Position,             nonatomic, AntPosition,          "AntPosition", Class that returns the sensory
                                                             position of the ant
class:Position,             nonatomic, EvalPosition,         "EvalPosition", Class that returns the
                                                             evaluated position of the ant
class:Position,             nonatomic, StartPosition,        "StartPosition", Class that returns the start
                                                             position of the robot
class:Position,             nonatomic, GoalPosition,         "GoalPosition", Class that returns the goal
                                                             position of the robot
class:Position,             nonatomic, ErrorPosition,        "ErrorPosition", Class that returns the error
                                                             between the goal and current positions
class:Orientation,          nonatomic, AntOrientation,       "AntOrientation", Class that returns the
                                                             orientation of the ant
class:Waypoints,            nonatomic, Mission,              "Mission", Class that returns the target
                                                             positions in the mission
class:Faults,               nonatomic, FaultMsg,             "FaultMsg", Class that returns the faults in
                                                             the system
bool,                       nonatomic, DemandStop,           "DemandStop", Stop mission message
bool,                       nonatomic, DemandModify,         "DemandModify", Modify mission message
bool,                       nonatomic, MotionInitialized,    "MotionInitialized", Motion initialization
                                                             message
bool,                       nonatomic, TargetRequest,        "TargetRequest", Target request message
bool,                       nonatomic, TargetReached,        "TargetReached", Target reached message
bool,                       nonatomic, WithinCorridor,       "WithinCorridor", Within corridor message
int,                        nonatomic, GaitType,             "GaitType", The type of gait
int,                        nonatomic, TargetTotal,          "TargetTotal", Total number of target
                                                             positions
int,                        nonatomic, TargetCurrent,        "TargetCurrent", Current target index
int,                        nonatomic, FSMid,                "FSMid", FSM client ID message
int,                        nonatomic, AntHeading,           "AntHeading", Heading angle of the robot
int,                        nonatomic, ErrorHeading,         "ErrorHeading", Error in the heading of the
                                                             robot
int,                        nonatomic, AntSlopeA,            "AntSlopeA", Surface inclination angle alpha
int,                        nonatomic, AntSlopeB,            "AntSlopeB", Surface inclination angle beta
```

Messages in the gusimplewhiteboardtypelist.tsl file