MOBILE PLATFORM CONTROL USING FUZZY-LOGIC AND WEBOTS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE FOR THE DEGREE OF

Bachelor of Technology

In

Mechanical Engineering

By

SURAJ.N.J

(ROLL.NUMBER: 10503039)

Department of Mechanical Engineering

National Institute of Technology

Rourkela

2008-09

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the project entitled, "Mobile Platform Control Using Fuzzy-Logic and WEBOTS" submitted by Sri Suraj.N.J in partial fulfilment of the requirements for the award of Bachelor of Technology, Rourkela (Deemed University) is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Date:

Dr.D.R.K.Parhi

Dept. of Mechanical Engineering

National Institute of Technology

Rourkela – 769008

India

National Institute of Technology

Rourkela

ACKNOWLEDGEMENT

I would like to articulate my deep gratitude to my project guide Dr.D.R.K.Parhi who has always been my motivation for carrying out the project.

It is my pleasure to refer Microsoft Word 2007 of which the compilation of this project would have been impossible.

An assemblage of this nature could never have been attempted without reference to and inspiration from the works of others whose details are mentioned in reference section. I acknowledge my indebtedness to all of them.

Last but not the least to all of my friends, who patiently extended all sorts of help for accomplishing this undertaking.

Date:

Suraj.N.J

Dept. of Mechanical Engineering

National Institute of Technology

Rourkela – 769008

India

CONTENTS

ABSTRACT

In this project, we study about the designing, controlling and successful working of robots under different environmental conditions and topography using WEBOTS simulator and try to optimise its functioning using Fuzzy-Logic. A robot carrying out a particular needed task has promising applications for the betterment of human society.

A well written code in WEBOTS simulator helps us to utilise the sensor information and integrate it with the robot's motor control to achieve the desired goal effectively. In order to synthesize the robot's controller, we rely on Fuzzy-Logic, which we show to be a powerful tool for the production of simple and effective solutions for our problem.

At the end, the performance of BOT performance with/without Fuzzy-Logic can be compared and used for further improvement.

Chapter 1

INTRODUCTION

1. Objective
2. Roots and Strategy

INTRODUCTION

We describe robot working under varied conditions and successfully carrying out its assigned task using the help of WEBOTS simulator and optimising its function using Fuzzy-Logic.

OBJECTIVE:

The main scientific objective of the project is the study of the novel ways of designing and effective execution of the task, based on Fuzzy-Logic.

ROOTS and STRATEGY:

Mobile Robotics is an emerging field of robotics that studies the behaviour of robots under dynamic and challenging conditions to achieve its goal.

Mobile Robotics successfully incorporates all the constraints that the robot experiences in its due course of operation and induces behaviour of self-thinking to the robot by harnessing the power of optimisation and intelligent techniques like Fuzzy-Logic, etc.

Fuzzy-Logic is used in system control and analysis design, because it shortens the time for engineering development and sometimes, in case of highly complex systems, is the only way to solve the problem. Fuzzy-Logic is based on the theory of fuzzy-sets, where an object's membership of a set is gradual rather than just member or not-a-member. It uses the whole interval of real numbers between zero (0 or False) and (1 or True) to develop logic as a basis for rules of inference.

Fuzzy-Logic is inspired by and is an approximation to human reasoning. It is governed by "Law of Excluded Middle", which states that "A system, in which propositions must be either true or false, but not both, uses a two-valued logic. As a consequence, what is not true is false and vice-versa." This calls for the introduction of "Membership Grade/Function – assigning a real number in the closed interval [0, 1] instead of {0} or {1}", which allows finer detail, such that the transition from Membership to Non-Membership is gradual rather than abrupt.

WEBOTS:

I did the simulation with help of **WEBOTS** version 5.10.0 which a 3D mobile robot simulator is allowing the users to simulate different types of mobile robots, including wheeled robots, legged robots and flying robots.

WEBOTS is professional mobile robot simulation software. It contains a rapid prototyping tool allowing the user to create 3D virtual worlds with physics properties, such as mass repartition, joints, friction coefficients, etc. The user can add simple inert objects or

active objects called mobile robots. Users can create complex virtual worlds and simulate their robots within these environments. A complete programming library is provided to allow users to program the robots (usually using the C, C++ or Java languages). From the controller programs, it is possible to read sensor values and send motor commands to robots. Resulting robot controllers can be transferred to real robots (Khepera robot with C controllers, Hemisson robot with BotStudio controllers, Aibo, LEGO Mindstorms, etc.).

WEBOTS is well suited for research and education projects related to mobile robotics. Many mobile robotics projects have been relying on WEBOTS for years in the following areas:

• Mobile robot prototyping (academic research, automotive industry, aeronautics, vacuum cleaner industry, toy industry, lobbyism, etc.)
• Multi-agent research (swarm intelligence, collaborative mobile robots groups, etc.)
• Adaptive behaviour research (Genetic evolution, neural networks, adaptive learning, AI, etc.).
• Mobile robotics teaching (robotics lectures, C/C++/Java programming lectures, robotics contest, etc.)

Chapter 2

LITERATURE SURVEY

LITERATURE SURVEY

The authors, **Liu, et.al.[1]** have attempted in this paper ,a fuzzy logic-based real-time navigation controller is described.This controller combines the path planning and trajectory following as an integrated and coordinated unit so that it executes maneuvers such as docking and obstacle avoidance on-line. Only a little information, which is easily obtained through a low-cost sonar system, is necessary and is always available. Tight coupling between sensor data and control actions provides the autonomous mobile robot with the adaptability necessary for coping with a dynamically changing world. There is no separate path planning to be per- formed. Driving mechanism reacts immediately to perceive Sensor data as the mobile robot navigates through the world.

In other paper, **Simon, et.al [2]** have described Mobile robots are mechanical devices capable of moving in an environment with a certain degree of autonomy. Autonomous navigation is associated with the availability of external sensors that capture information from the environment through visual images, or through distance or proximity measurements. The most common sensors are distance sensors (ultrasonic, laser, etc.) capable of detecting obstacles and of measuring the distance to walls close to the robot path. When advanced autonomous robots navigate within indoor environments (industrial or civil buildings), they have to be endowed with the ability to move through corridors, to follow walls, to turn corners and to enter open areas of the rooms.

As regards the corridor and wall-following navigation problem, some control algorithms based on artificial vision have been proposed. In one, image processing is used to detect perspective lines to guide the robot along the center axis of the corridor. In other, two lateral cameras mounted on the robot are used, and the optical flow is computed to compare the apparent image velocity on both cameras in order to control robot motion. In other, one camera is used to drive the robot along the corridor axis or to follow a wall, by using optic flow computation and its temporal derivatives. In other, a globally stable control algorithm for wall-following based on incremental encoders and one sonar sensor is developed. In other, a theoretical model of a fuzzy based reactive controller for a non-holonomic mobile robot is developed. In other, an ultrasonic sensor is used to steer an autonomous robot along a concrete path using its edged as a continuous landmark. In other, a mobile robot control law for corridor navigation and wall following based on sensor and odometric sensorial information is proposed.

Kumar, et.al [3] have gone to explain that the mobile robot is a small four-wheeled mobile platform, which was controlled by a micro-controller. The robot could sense its surroundings with the aid of various electronic sensors while mechanical actuators

were used to move it around. Robot behaviour was determined by the program, which was loaded to the microcontroller. In that way, it could be used as a general robotics experimental platform. The autonomous mobile robot was designed and built in order to perform various navigation algorithms. The design consisted of two main sections: Electronic analysis of the various robot sensors and Programming techniques used to interface the sensors with the robot's microcontroller. In this paper it's shown that the path-guiding robot with IR sensors and obstacle detection is using IR proximity sensors. The predefined path is having varied turns, the fuzzy reasoning take care of speed to keep mobile robot in the defined path. The results are proved experimentally and the surface viewer graph is obtained from the Mat Lab.

This paper co-authored by **Castillo, et.al [4]** addresses the problem of trajectory tracking control in an autonomous, wheeled, mobile robot of unicycle type using Fuzzy Logic. The Fuzzy Logic Control (FLC) is based on a backstepping approach to ensure asymptotic stabilization of the robot's position and orientation around the desired trajectory, taking into account the kinematics and dynamics of the vehicle. We use the Mamdani inference system to construct a controller, with nine IF-THEN rules and the centroid of area method as our deffuzification strategy where the input torques and velocities are considered as linguistic variables. The performance of this FLC is illustrated in a simulation study.

The Tagaki-Sugeno approach is the most commonly used fuzzy logic model in the tracking control problem of autonomous vehicles.

One of the long standing challenging aspect in mobile robotics, which has been addressed here by **Fatmi, et.al [5]** is the ability to navigate autonomously, avoiding modeled and unmodeled obstacles especially in crowded and unpredictably changing environment. A successful way of structuring the navigation task in order to deal with the problem is within behavior based navigation approaches. In this study, Issues of individual behavior design and action coordination of the behaviors will be addressed using fuzzy logic. A layered approach is employed in this work in which a supervision layer based on the context makes a decision as to which behavior(s) to process (activate) rather than processing all behavior(s) and then blending the appropriate ones, as a result time and computational resources are saved.

This paper presented by **Malhotra, et.al [6]** presents the design of a mobile robot for obstacle avoidance in an environment about which no a-priori information is available and which consists of static as well as moving obstacles. The paper concerns itself with the design of a fuzzy brain for the mobile robot, its integration into the control system and the sensor system used for the detection of obstacles in its workspace. The obstacle avoidance strategy of the robot is based on the artificial potential field method. A fuzzy logic based system is used to implement this strategy since it reduces the computational effort required in the implementation of the artificial potential field method. An algorithm (intelligent obstacle avoidance algorithm) is proposed to

integrate the fuzzy system into the main control system for the mobile robot. The system described above is being tested by simulation and subsequently will also be tested on an actual mobile robot being developed.

The paper co-authored by **Singh, et.al [7]** follows an approach to robot control where desirable traits are expressed as quantitative preferences defined over the set of all possible control actions from the perspective of the goal associated with that behavior. For example, a behavior for avoiding obstacles could map configurations of sonar readings that correspond to the presence of an obstacle on the left of the robot into a function that prefers actions that steer the robot to the right.The paper calculates the desirability of a control by using only 1 level of estimation where the results of the control are used to calculate its desirability. We can extend it to include a sequence of controls so that we can look at more future conditions to take present action. This is much like the game tree techniques used by AI based computer programs.The other extension we suggest is to model the inputs from sensors as fuzzy variables which would take into account the practical noisy, time dependent nature of most sensors.

This paper introduced by **Ramos, et.al [8]** talks about a fuzzy decision-making algorithm for robot behavior coordination. The algorithm belongs to the arbitration class of behavior coordination mechanisms, under which only one behavior is running at a time.However, it is possible to use a hierarchical decision mechanism for hierarchical behaviors without interference between hierarchical levels.With this fuzzy decision method it is possible to represent a specific model of the world where the robot evolves. This algorithm consists of defining a set of behaviors, a set of world states, a cost function for behaviors, a set of goals, and a set of constraints. For each behavior and actual world state pair, a cost function is computed. The cost of each pair is evaluated by the overall goals. Goals and constraints are aggregated using a fuzzy operator and the optimal choice is the behavior with the maximum resulting value. This algorithm was tested with success in realistic simulations of a goalkeeper soccer robot.

This work done by **Busquets, et.al [9]** explores the use of bidding mechanisms to coordinate the actions requested by a group of agents in charge of achieving the task of guiding a root towards a specified target in an unknown environment. This approach is based on a fuzzy approach to landmark-based navigation.

Outdoor navigation in unknown environments is still a difficult open problem in the field of robotics. Existing approaches assume that an appropriately detailed and accurate metric map can be got through sensing the environment. However, most of these approaches rely on odometry sensors which can be very imprecise and lead to many errors.Our approach considers using only visual information. The robot must be equipped with a visual system capable of recognising visual salient obejcts, and use them for mapping and navigation tasks.

The agents' theory offers flexibility for solution of problems whose environment is dynamic and imprecise. The use of the computational intelligence together with the agents' theory seems to be a natural way of providing an agent with intelligence. In this paper we, **Figueiredo, et.al [10]** describe the use of intelligent agents, whose intelligence is based on a fuzzy logic system, applied to the control of a robot, simulated by the Khepera simulator. Fuzzy Logic Systems have demonstrated, through the numerous applications in the area, to be an effective procedure for control problems. The attitude to be taken at each moment by an agent is defined by a set of fuzzy rules based upon the robot position, its sensor values, distance and angle relative to the target. To prevent the robot from getting stuck by some obstacles, a path memory system was created, forcing the robot to seek new alternatives when it gets trapped. The results obtained demonstrate a successful combination of Computational Intelligence and the Theory of Agents in a control system with ability to avoid deadlock situations.

This paper written by **Michel [12]** presents Webots: a realistic mobile robot simulator allowing a straightforward transfer to real robots. The simulator currently supports the Khepera mobile robot and a number of extension turrets. Both real and simulated robots can be programmed in C language using the same Khepera API, making the source code of a robot controller compatible between the simulator and the real robot. Sensor modelling for 1D and 2D cameras as well as visualisation and environment modelling are based upon the OpenGL 3D rendering library. A file format based on an extension of VRML97, used to model the environments and the robots, allows virtual robots to move autonomously on the Internet and enter the real world. Current applications include robot vision, artificial life games, robot learning, etc.

This paper presents a new method for behavior fusion control of a mobile robot in uncertain environments. Using behavior fusion by fuzzy logic, a mobile robot is able to directly execute its motion according to range information about environments, acquired by ultrasonic sensors, without the need for trajectory planning. Based on low-level behavior control, **Wei, et.al. [13]** construct an efficient strategy for integrating high-level global planning for robot motion can be formulated, since, in most applications, some information on environments is prior knowledge. A global planner, therefore, only needs to generate some subgoal positions rather than exact geometric paths. Because such subgoals can be easily removed from or added into the planner, this strategy reduces computational time for global planning and is flexible for replanning in dynamic environments. Simulation results demonstrate that the proposed strategy can be applied to robot motion in complex and dynamic environments.

Chapter 3

WORK ANALYSIS

1. **My first bot**
2. **Environment**
3. **A simple controller**
4. **Implementation of Fuzzy-Logic in Webots**

3.1 My first BOT:

As a first introduction, we are going to simulate a very simple robot made up of a cylinder, two wheels and two infrared sensors. A program performing obstacle avoidance inspired from **Braitenberg's algorithm** controls the robot. It evolves in a simple environment surrounded by a wall, which contains some obstacles to avoid.

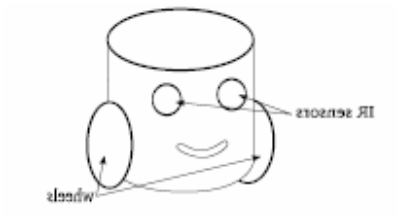


Figure 3.1: My First-Bot

3.2 Environment:

This very first simulated world is as simple as possible. It includes a floor, 4 obstacles and a surrounding wall to avoid that the robot escapes. This wall is modelled using an Extrusion node.

First, launch WEBOTS and stop the current running simulation by pressing the **Stop** button. Go to the **File** menu, **new** item to create a new world. This can also be achieved through the **New** button, or the keyboard shortcut indicated in the **File** menu. Then open the scene tree window from the **Scene Tree...** item in the **Edit** menu. This can also be achieved by double-clicking in the 3D world. Let us start by changing the lighting of the scene:

1. Select the **Point Light node**, and click on the + just in front of it. You can now see the different fields of the **Point Light** node. Select **ambient Intensity** and enter 0.6 as a value, then select intensity and enter 0.6, then, select location and enter [0.75 0.5 0.5] as values. Press return.

2. Select the **Point Light** node, copy and paste it. Open this new **Point Light** node and type [-0.5 0.5 0.35] in the location field.

3. Repeat this paste operation twice again with [0.45 0.5 -0.5] in the location field of the third **Point Light** node, and [-0.5 0.5 -0.35] in the location field of the fourth and last **Point Light** node.

4. The scene is now better lit. Open the **Preferences...** from the **Edit** menu, select the **Rendering** tab and check the **Display lights** option. Click on the **OK** button to leave the preferences and check that the light sources are now visible in the scene. Try the different mouse buttons, including the mouse wheel if any, and drag the mouse in the scene to navigate and observe the location of the light sources.

Secondly, let us create the wall:

1. Select the last **Transform** node in the scene tree window (which is the floor) and click on the **insert after** button.

2. Choose a **Solid** node.

3. Open this newly created **Solid** node from the + sign and type "**wall**" in its name field.

4. Select the children field and **Insert after** a **Shape** node.

5. Open this Shape, select its appearance field and create an Appearance node from the **New node** button. Use the same technique to create a Material node in the material field of the **Appearance** node. Select the **diffuseColor** field of the **Material** node and choose a colour to define the colour of the wall. Let us make it light brown. In order to make your object change its colour depending on its illumination, select the **specularColour** field of the **Material** node and choose a colour to define the colour of the illuminated wall. Let us use an even lighter brown to reflect the effect of the light.

6. Similarly it also is possible to easily modify the colours of the ground. To do so you will have to modify the two colour fields of the last **Transform** node, the one corresponding to the ground, which are located in **the children / Shape / geometry / Colour** node. In our examples we have changed it to a black and white grid.

7. Now create an **Extrusion** node in the **geometry** field of the **Shape**.

8. Set the convex field to FALSE. Then, set the wall corner coordinates in the **crossSection** field as shown in. You will have to re-enter the first point (0) at the last position (10) to complete the last face of the extrusion.

9. In the spine field, write that the wall ranges between 0 and 0.1 along the Y axis (instead of the 0 and 1 default values).

10. As we want to prevent our robot to pass through the walls like a ghost, we have to define the **boundingObject** field of the wall. Bounding objects cannot use complex geometry objects. They are limited to box, cylinder and spheres primitives. Hence, we will have to create four boxes (representing the four walls) to define the **boundingobject** of the surrounding wall. Select the **boundingObject** field of the wall and create a **Group** node that will contain the four walls. In this **Group**, insert a **Transform** node as **children**. Create a Shape as the unique children of the Transform. Create a
Material in the node **Appearance** and set both of its **diffuseColor** and **specularColour** to white. This will be useful later, when the robot will have to detect the obstacles because the detection of the sensors is based on these colours. Now create a **Box** as **geometry** for this **Shape** node. Set the size of the Box to [0.01 0.1 1], so
that it matches the size of a wall. Set the translation field of the **Transform** node to [0.495 0.05 0], so that it matches the position of a wall.

11. Now, close this **Transform**, copy and paste it as the second children of the list.

Instead of creating a new **Shape** for this object, reuse the **Shape** you created for the first bounding object. To do so, go back to the **Transform** node of the previous object, open the **children** node, click on the **Shape** node and you will see on the right hand side of the window that you can enter a **DEF** name. Write **WALL_SHAPE** as a **DEF** name and return to the children of the second bounding object. First **Delete** the **Shape** contained in it and create a **New node** inside it. However, in the **Create new node** dialog, you will now be able to use the **WALL_SHAPE** you just defined. Select this item and click **OK**. Set the translation field of the new node to [-0.495 0.05 0], so that it matches the opposite wall. Repeat this operation with the two
remaining walls and set their rotation fields to [0 1 0 1.57] so that they match the orientation of the corresponding walls. You also have to edit their translation field as well, so that they match the position of the corresponding walls.

12. Close the tree editor, save your file as "**my_mybot.wbt**" and look at the result.

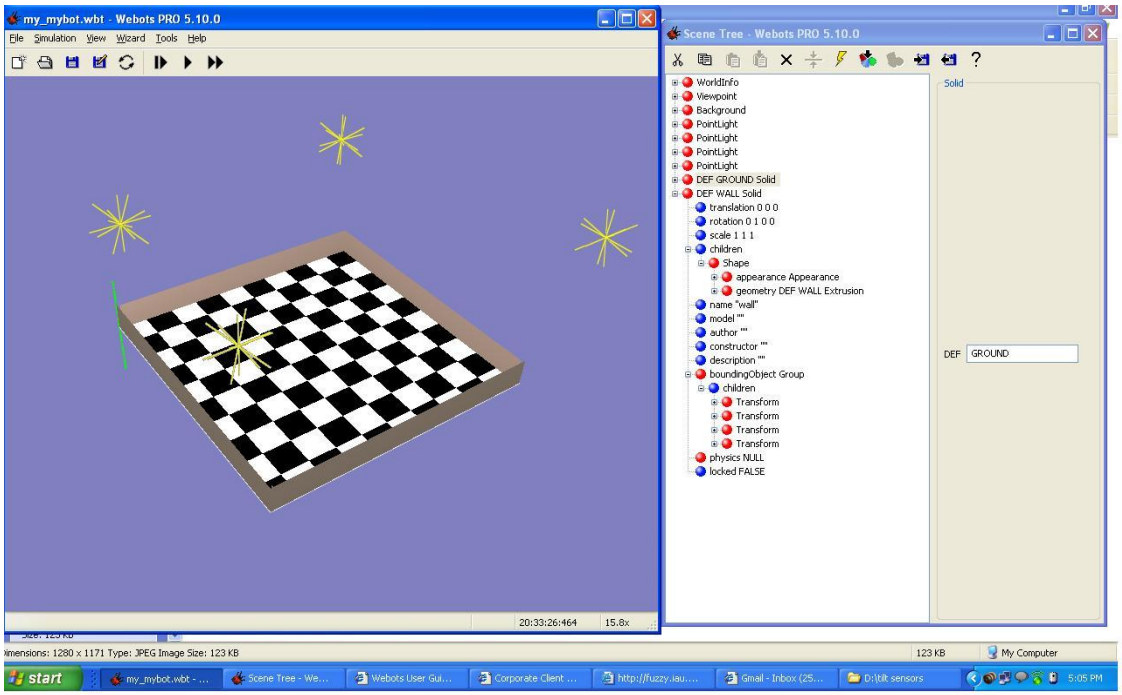


Figure 3.2: My WEBOT world (illuminated by 4 lights)

Thirdly, let us create the obstacles:

1. Select the last **Solid** node in the scene tree window (which is the wall) and click on the insert after button.

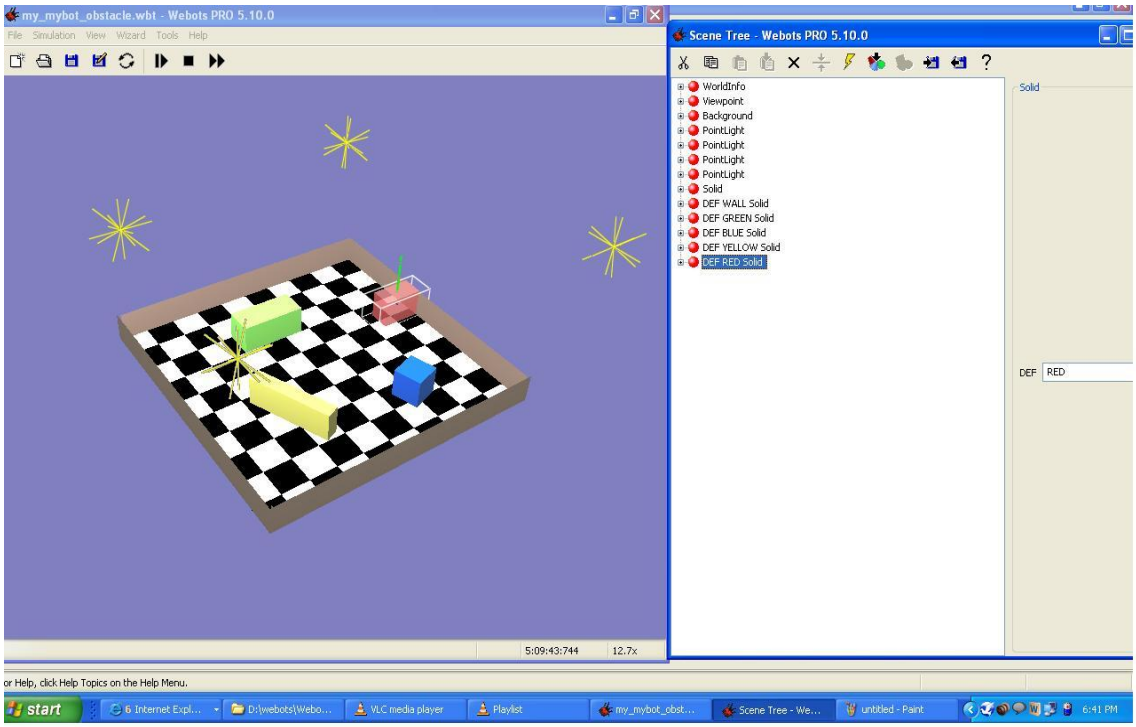


Figure 3.3: My WEBOT world with Obstacles

2. Choose a **Solid** node.

3. Open this newly created **Solid** node from the + sign and type "**green box**" in its name field.

4. Using the same technique as for the wall add first a Shape, then an **Appearance** and a **Material**. For the colour, let us make it green with a lighter green for the illuminated parts.

5. Now create a **Box** node in the geometry field of the Shape and set its size to [0.23 0.1 0.1]. Set the **DEF** name of this geometry to **BOX0**.

6. To create the **boundingObject** of this object, create a **Shape** node and reuse the previous **DEF** for the geometry. As for the wall, create also an **Appearance** and a **Material** node and set the two colours to white.

7. Finally set the **translation** field to [-0.05 0.05 -0.25] but let its **rotation** field to the standard values.

8. Now repeat these steps to create the three remaining obstacles. First create the one called "**blue box**" which has a geometry called **BOX1** of [0.1 0.1 0.1], a **translation** of [0.2 0.05 0.27] and a **rotation** of [0 1 0 0.31]. Then create the one called "**yellow box**" which has a **geometry** called **BOX2** of [0.05 0.1 0.3], a translation of [-0.2 0.05 0.15] and a **rotation** of [0 1 0 0.4]. Finally create the one called "**red box**" which has a **geometry** called **BOX3** of [0.15 0.1 0.08], a **translation** of [0.42 0.05 -0.1] and a standard **rotation**. For all these objects, set their colours accordingly with their names.

Robot

This subsection describes how to model the ***MyBot*** robot as a **DifferentialWheels** node containing several children: a **Transform** node for the body, two **Solid** nodes for the wheels, two **DistanceSensor** nodes for the infra-red sensors and a **Shape** node with a texture. The origin and the axis of the coordinate system of the robot and its dimensions are shown.

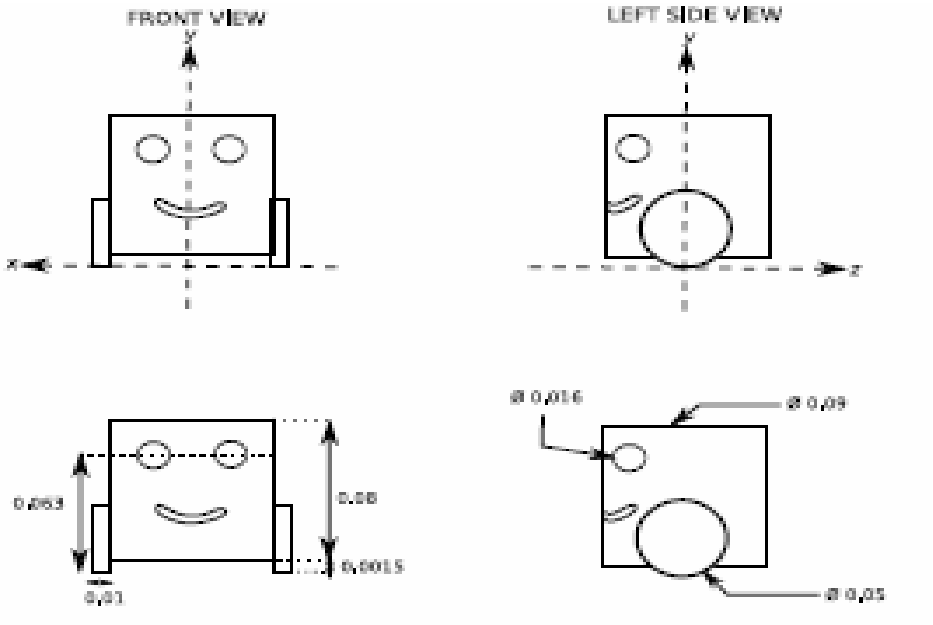


Figure 3.4: Dimensions of My First Bot

To model the body of the robot:

1. Open the scene tree window.

2. Select the last **Solid** node.

3. **Insert after** a **DifferentialWheels** node, set its name to "**MyBot**".

4. In the children field, first introduce a Transform node that will contain a shape with a cylinder. In the new children field, **Insert after** a Shape node. Choose a colour, as described previously. In the geometry field, **insert** a Cylinder node. Set the height field of the cylinder to 0.08 and the radius one to 0.045. Set the **DEF** name of the **geometry** to **BODY**, so that we will be able to reuse it later. Now set the **translation.**

To model the left wheel of the robot:

1. Select the Transform node corresponding to the body of the robot and **Insert after** a **Solid** node in order to model the left wheel. Type "**left wheel**" in the name field, so that this **Solid** node is recognized as the left wheel of the robot and will rotate according to the motor command.

2. The axis of rotation of the wheel is *x*. The wheel will be made of a Cylinder rotated of *pi/2* radians around the *z* axis. To obtain proper movement of the wheel, you must pay attention not to confuse these two rotations. Consequently, you must add a **Transform** node to the children of the **Solid** node.

3. After adding this **Transform** node, introduce inside it a **Shape** with a Cylinder in its **geometry** field. Don't forget to set an appearance as explained previously. The dimensions of the cylinder should be 0.01 for the height and 0.025 for the radius.
Set the **rotation** to [0 0 1 1.57]. Pay attention to the sign of the rotation; if it is wrong, the wheel will turn in the wrong direction.

4. In the **Solid** node, set the **translation** to [-0.045 0.025 0] to position the left wheel, and set the rotation of the wheel around the *x* axis:
[1 0 0 0].

6. Close the tree window, look at the world and save it. Use the navigation buttons to change the point of view.

To model the right wheel of the robot:

1. Select the left wheel Solid node and **insert after** another Solid node. Type "**right wheel**" in the name field. Set the translation to [0.045 0.025 0] and the rotation to [1 0 0 0].

2. In the children, **Insert after** USE WHEEL. Press **Return**, close the tree window and save the file. You can examine your robot in the world editor, move it and zoom on it.

The two infra-red sensors are defined as two cylinders on the front of the robot body. Their diameter is 0.016 m and their height is 0.004 m. You must position these sensors properly so that the sensor rays point in the right direction, toward the front of the robot.

1. In the children of the **DifferentialWheels** node, **insert after** a **DistanceSensor** node.

2. Type the name "ir0". It will be used by the controller program.

3. Let us attach a cylinder shape to this sensor: In the children list of the **DistanceSensor** node, **Insert after** a **Transform** node. Give a **DEF** name to it: **INFRARED**, which you will use for the second IR sensor.

4. In the children of the **Transform** node, **insert after** a **Shape** node. Define an **appearance** and **insert** a Cylinder in the **geometry** field. Type 0.004 for the height and 0.008 for the radius.

5. Set the rotation for the **Transform** node to [0 0 1 1.57] to adjust the orientation of the cylinder.

6. In the **DistanceSensor** node, set the translation to position the sensor and its ray: [-0.02 0.063 -0.042]. In the **File** menu, **Preferences**, **Rendering**, check the **Display sensor rays** box. In order to have the ray directed toward the front of the robot, you must set the rotation to [0 1 0 2.07].

7. In the **DistanceSensor** node, you must introduce some values of distance measurements of the sensors to the **lookupTable** field, according to and these values are:

Lookup Table [0 1024 0,
0.05 1024 0,
0.15 0 0]

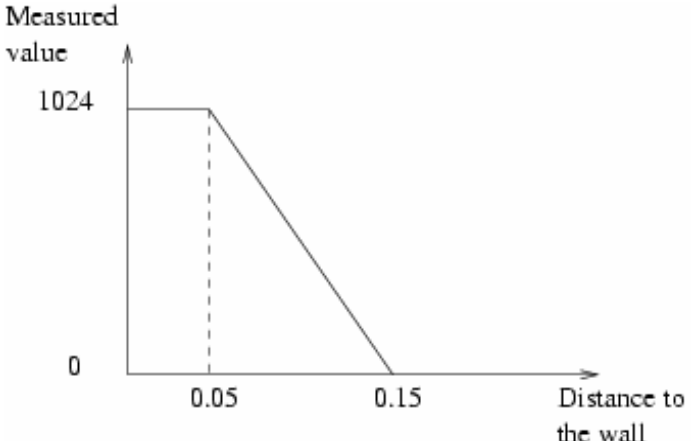


Figure 3.5: Distance measurements of the *MyBot* sensors.

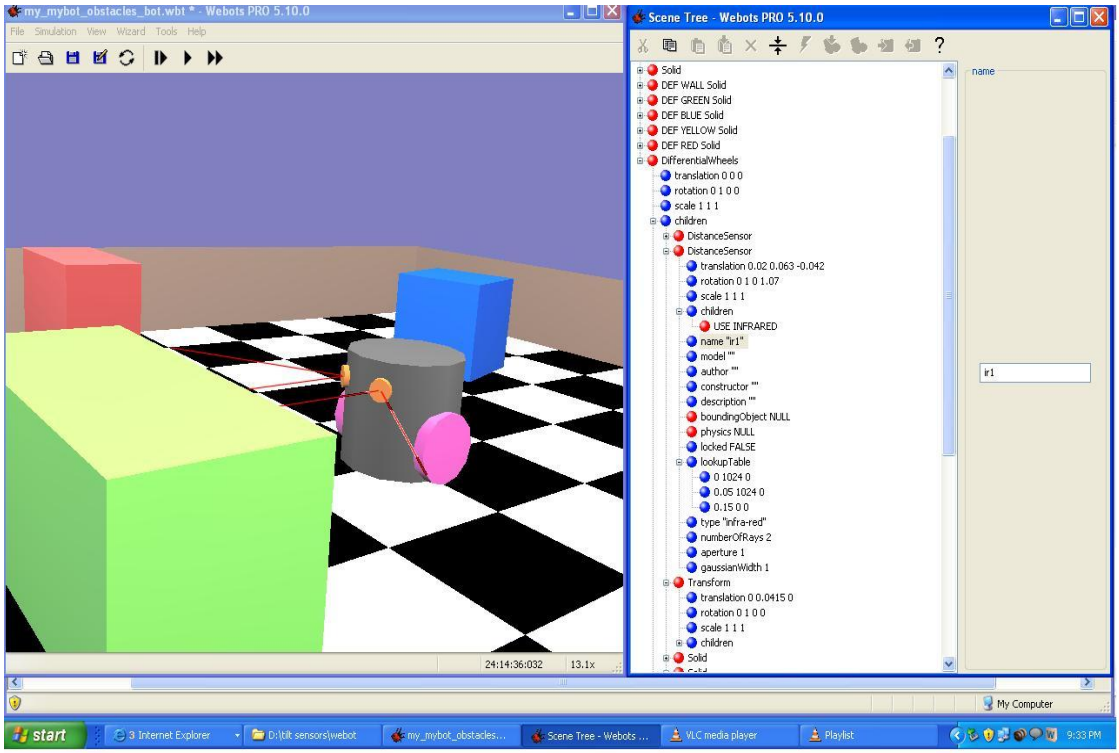


Figure 3.6: My First Bot with 2 IR Sensors

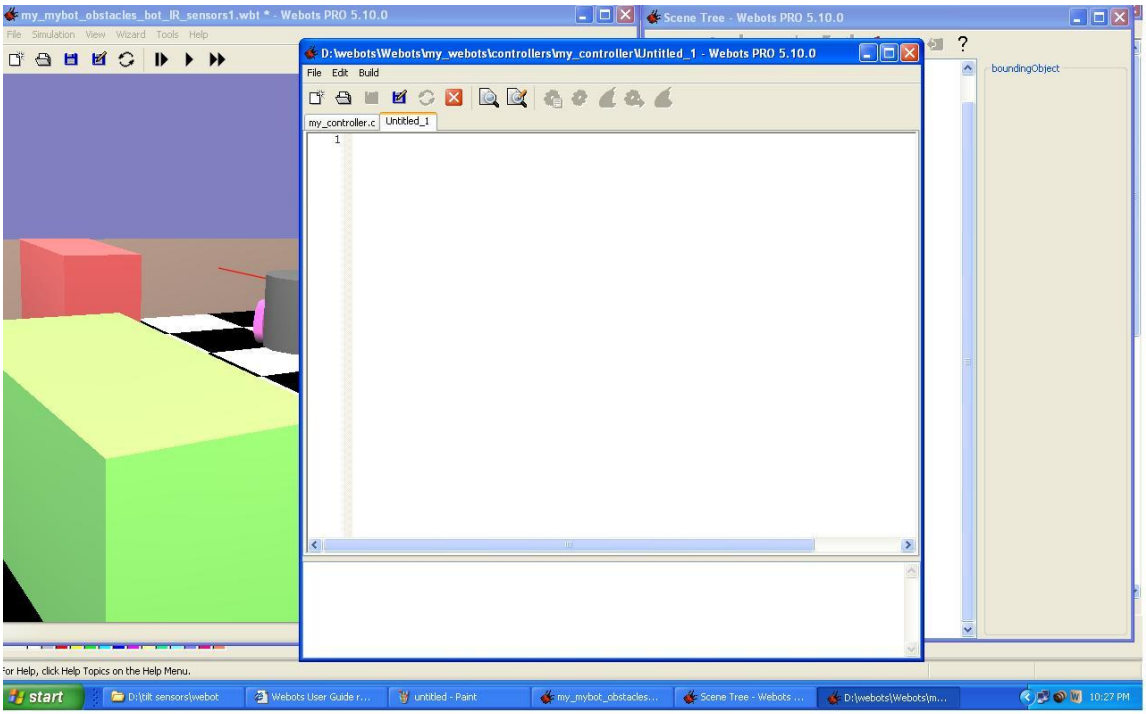


Figure 3.7: Controller Program Space

3.3 CONTROLLER :

Synchronous Controllers:

They are recommended for robust control.

Asynchronous Controllers:

They are recommended for running robot competitions where computer resources are limited, or networked simulations involving several robots distributed over a computer network with an unpredictable delay, (like Internet) (also, if computationally expensive controller).

The above 2 controllers can be used selectively for Synchronous and Asynchronous Robots.

SIMPLE CONTROLLER PROGRAM

This controller is very simple. The controller program simply reads the sensor values and sets the two motors' speeds, in such a way that **MyBot** avoids the obstacles.

Below is the source code for the **mybot_simple.c** controller:

```
#include <device/robot.h>
#include <device/differential_wheels.h>
#include <device/distance_sensor.h>

#define SPEED 60
#define TIME_STEP 64

static void reset(void);
static int run(int);

static DeviceTag ir0, ir1;

static void reset(void)
{
ir0 = robot_get_device("ir0");
ir1 = robot_get_device("ir1");
distance_sensor_enable(ir0, TIME_STEP);
distance_sensor_enable(ir1, TIME_STEP);
return;
}

static int run(int ms)
{
short left_speed, right_speed;
unsigned short ir0_value, ir1_value;

ir0_value = distance_sensor_get_value(ir0);
ir1_value = distance_sensor_get_value(ir1);

if (ir1_value > 500) {
if (ir0_value > 500) {

left_speed = -SPEED;
right_speed = -SPEED / 2;
} else {
left_speed = -ir1_value / 10;
right_speed = (ir0_value / 10) + 5;

} else if (ir0_value > 500) {
left_speed = (ir1_value / 10) + 5;
right_speed = -ir0_value / 10;
} else {
```

```
left_speed = SPEED;
right_speed = SPEED;
}

differential_wheels_set_speed(left_speed, right_speed);
return TIME_STEP;
}
int main()
{
robot_live(reset);
robot_run(run);
return 0;
}
```

3.4 APPLICATION OF FUZZY-LOGIC USING MATLAB:

For efficient avoidance of obstacles, let's define 3 inputs and expect 2 desired outputs.

3 Chosen Inputs: (got from distance-measuring sensors)

a. Front Obstacle Distance (FOD)
b. Left Obstacle Distance (LOD)
c. Right Obstacle Distance (ROD)

2 Desired Ouputs:

a. Left Wheel Velocity (LWV)
b. Right Wheel Velocity (RWV)

IMPLEMENTATION OF FUZZY-SYSTEM:

This job is initially carried out by MATLAB FUZZY-LOGIC TOOLBOX and later by MATLAB written program.

Steps:

a. Type "anfisedit" in command window of MATLAB.
b. Click "File" , "New FIS", Mamdani
c. The above FIS Editor is used to define our inputs and outputs.
d. Click "Edit", "Add Variables", "Input / Output".
e. Membership function is chosen according to our need, viz. trapezoidal, triangular, gaussian, sigma, etc.
f. Defuzzification is done by Centroid Method.

DEFINITION OF INPUTS & OUTPUTS:

1. INPUTS : FOD, LOD, ROD can be Far (F) or Near (N)
2. OUTPUTS: LWV, RWV can be Fast (f) or Slow (s).

RULES FOR OPERATION:

Sl.No	FOD	LOD	ROD	LWV	RWV
1	F	F	F	F	f
2	F	F	N	S	f
3	F	N	F	F	s
4	F	N	N	F	f
5	N	F	F	S	f
6	N	F	N	S	f
7	N	N	F	F	s
8	N	N	N	S	s

The above logic is fed into Fuzzy-Controller via RULE-EDITOR.

INCORPORATION OF FUZZY-LOGIC INTO WEBOTS SIMULATOR:

The following MATLAB written Code incorporates Fuzzy-Logic into Main-Controller Program.

There are 2 ways to incorporate Fuzzy-logic into Webots and test:

1. Include the Fuzzy-Logic Code into Webots controller Code.
2. Get the values (Distance-sensor) values from Webots and include into Fuzzy-logic Code and plot it in Mamdani Matlab controller FIS Editor.

The Strategy adopted here in this thesis is the SECOND Way.

MATLAB FUZZY-LOGIC CODE:

PROGRAM TO INCORPORATE FUZZY-LOGIC INTO WEBOTS:

```
%[System]

Name='fuzzy-logic';

Type='mamdani';

Version=2.0;

NumInputs=3;

NumOutputs=2;

NumRules=8;

AndMethod='min';

OrMethod='max';

ImpMethod='min';

AggMethod='max';

DefuzzMethod='centroid';

x = 0:0.1:6;


%Rules

if (FOD == 'F' & LOD == 'F'& ROD == 'F')
```

```
LWV = 'f';
RWV = 'f';
end

if (FOD == 'F' & LOD == 'F'& ROD == 'N')
LWV = 's';
RWV = 'f';
end

if (FOD == 'F' & LOD == 'N'& ROD == 'F')
LWV = 'f';
RWV = 's';
end

if (FOD == 'F' & LOD == 'N'& ROD == 'N')
LWV = 'f';
RWV = 'f';
end

if (FOD == 'N' & LOD == 'F'& ROD == 'F')
LWV = 's';
RWV = 'f';
end

if (FOD == 'N' & LOD == 'N'& ROD == 'N')
LWV = 's';
```

```
RWV = 's';
end

if (FOD == 'N' & LOD == 'F'& ROD == 'N')
LWV = 's';
RWV = 'f';
end

if (FOD == 'N' & LOD == 'N'& ROD == 'F')
LWV = 'f';
RWV = 's';
end


%[Input1]
Name='FOD'
Range=[0 6];
NumMFs=2;
if ( FOD == 'F' )
MF1= trimf ( x, [2 4 6] );
end
if ( FOD == 'N' )
MF2= trimf ( x, [0 2 4] );
end
plot(MF1,MF2), title('FOD'),
xlabel('FOD values'),
```

```
ylabel('Membership function value'),grid
out1  = defuzz(x,MF1,'centroid')
out11 = defuzz(x,MF2,'centroid')
out_1 = min(out1, out11)

%[Input2]
Name='LOD'
Range=[0 6];
NumMFs=2;
if ( LOD == 'F' )
MF1= trimf ( x, [3.5 4.5 5.5] );
end
if ( LOD == 'N' )
MF2= trimf ( x, [0.5 2.5 4.5] );
end
plot(MF1,MF2), title('LOD'),
xlabel('LOD values'),
ylabel('Membership function value'),grid
out2  = defuzz(x,MF1,'centroid')
out22 = defuzz(x,MF2,'centroid')
out_2 = min(out2, out22)

%[Input3]
Name='ROD'
Range=[0 6];
NumMFs=2;
```

```
if ( ROD == 'F' )
MF1= trimf ( x, [0.6 3.2 4.4] );
end
if ( ROD == 'N' )
MF2= trimf ( x, [3.6 5 6] );
end
plot(MF1,MF2), title('ROD'),
xlabel('ROD values'),
ylabel('Membership function value'),grid
out3  = defuzz(x,MF1,'centroid')
out33 = defuzz(x,MF2,'centroid')
out_3 = min(out3, out33)

out_final = min ( out_1, out_2, out_3 )

%[Output1]
Name='LWV'
Range=[0 4];
NumMFs=2;
if ( LWV == 'f' )
MF1= trimf ( x, [2 3.5 4] );
end
if ( LWV == 's' )
MF2= trimf ( x, [0 1 2] );
end
plot(MF1,MF2), title('LWV'),
```

```
xlabel('LWV values'),
ylabel('Membership function value'),grid
outf1  = defuzz(x,MF1,'centroid')
outf11 = defuzz(x,MF2,'centroid')
outf_1 = min(outf1, outf11)
outf_1net = outf_1./out_final

%[Output2]
Name='RWV'
Range=[0 4];
NumMFs=2;
if ( RWV == 'f' )
MF1= trimf ( x, [1.25 2.5 3.6] );
end
if ( RWV == 's' )
MF2= trimf ( x, [0 1.75 3] );
end
plot(MF1,MF2), title('RWV'),
xlabel('RWV values'),
ylabel('Membership function value'),grid
outf2  = defuzz(x,MF1,'centroid')
outf22 = defuzz(x,MF2,'centroid')
outf_2 = min(outf2, outf22)
outf_2net = outf_2./out_final

% [Rules]
```

% 1 1 2, 1 1 (1) : 1

% 1 1 1, 2 1 (1) : 1

% 1 2 2, 1 2 (1) : 1

% 1 2 1, 1 1 (1) : 1

% 2 1 2, 2 1 (1) : 1

% 2 1 1, 2 1 (1) : 1

% 2 2 2, 1 2 (1) : 1

% 2 2 1, 2 2 (1) : 1

% Defuzzification - Centroid

%out = defuzz(x,MF,'centroid')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Chapter 4

RESULTS

Output with Figures from Simulation

PATH TRAVELLED BY BOT WITHOUT FUZZY LOGIC IMPLEMENTATION:

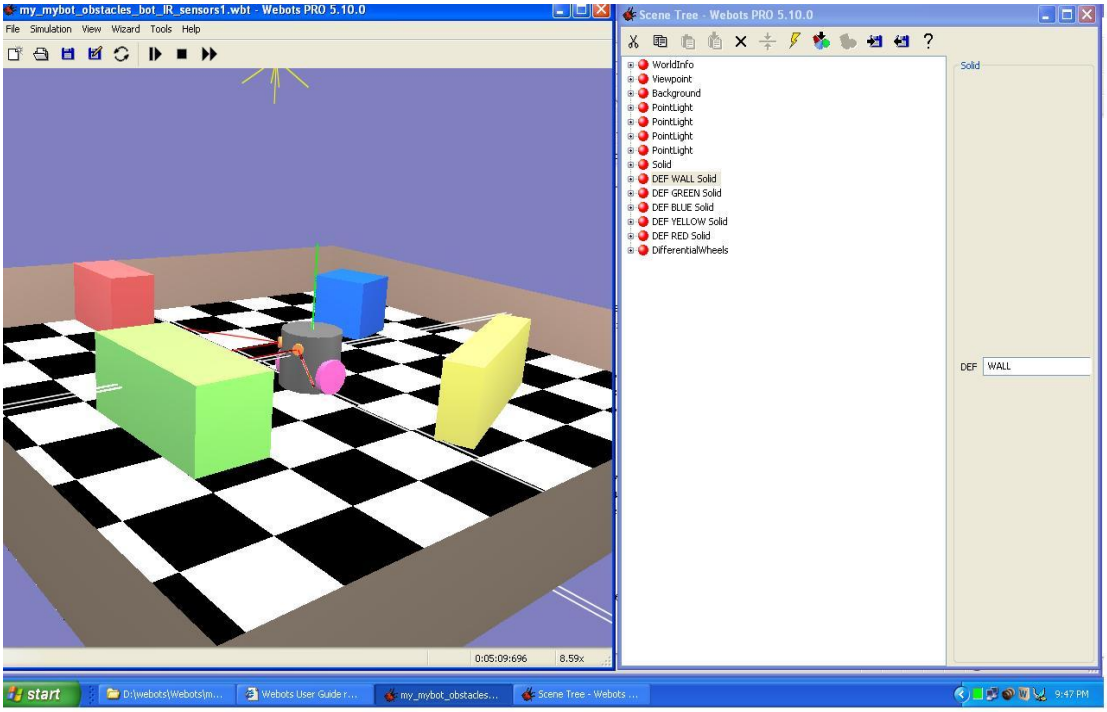


Figure 4.1: My Bot before avoiding Obstacles (in its original initial position)

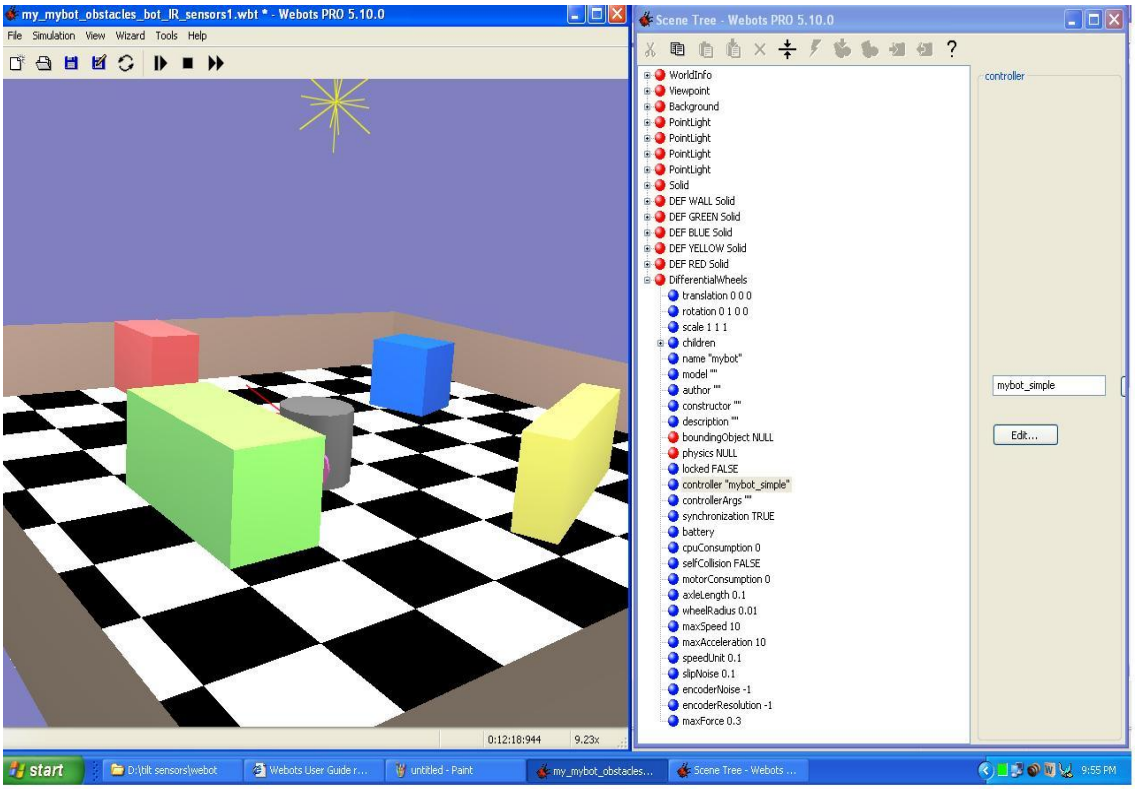


Figure 4.2: My Bot avoiding the First Obstacle (Green)

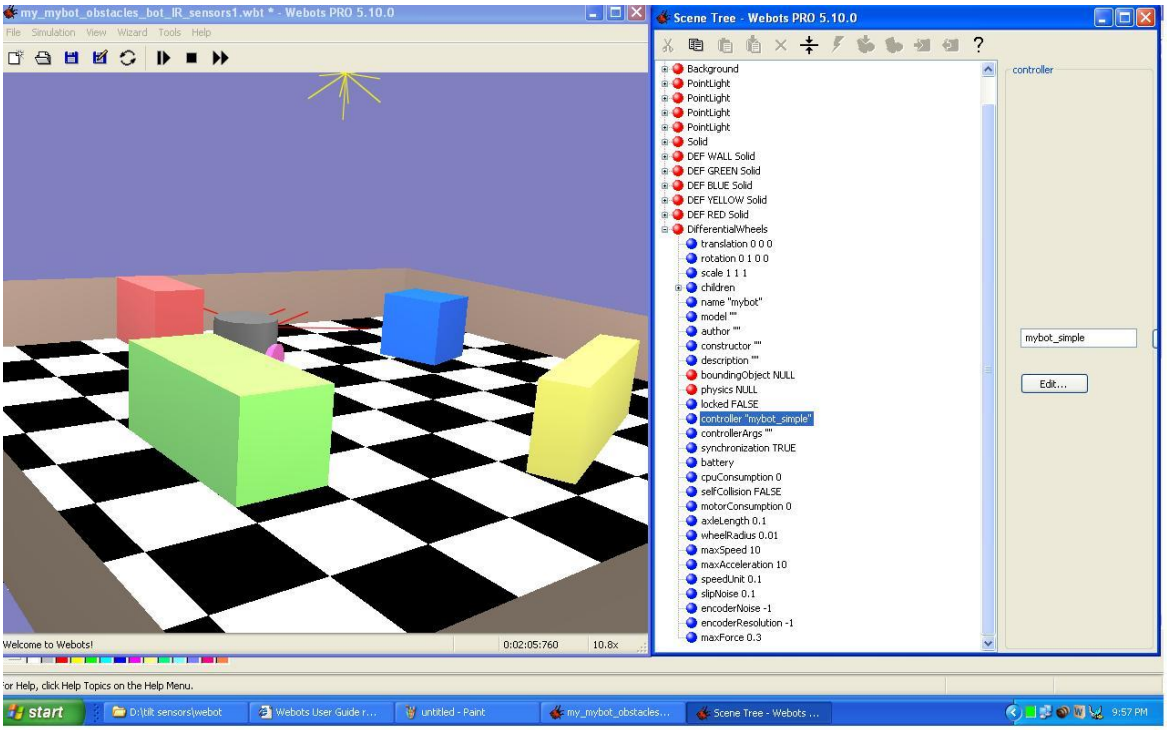


Figure 4.3: My Bot avoiding the Second Obstacle (Red)

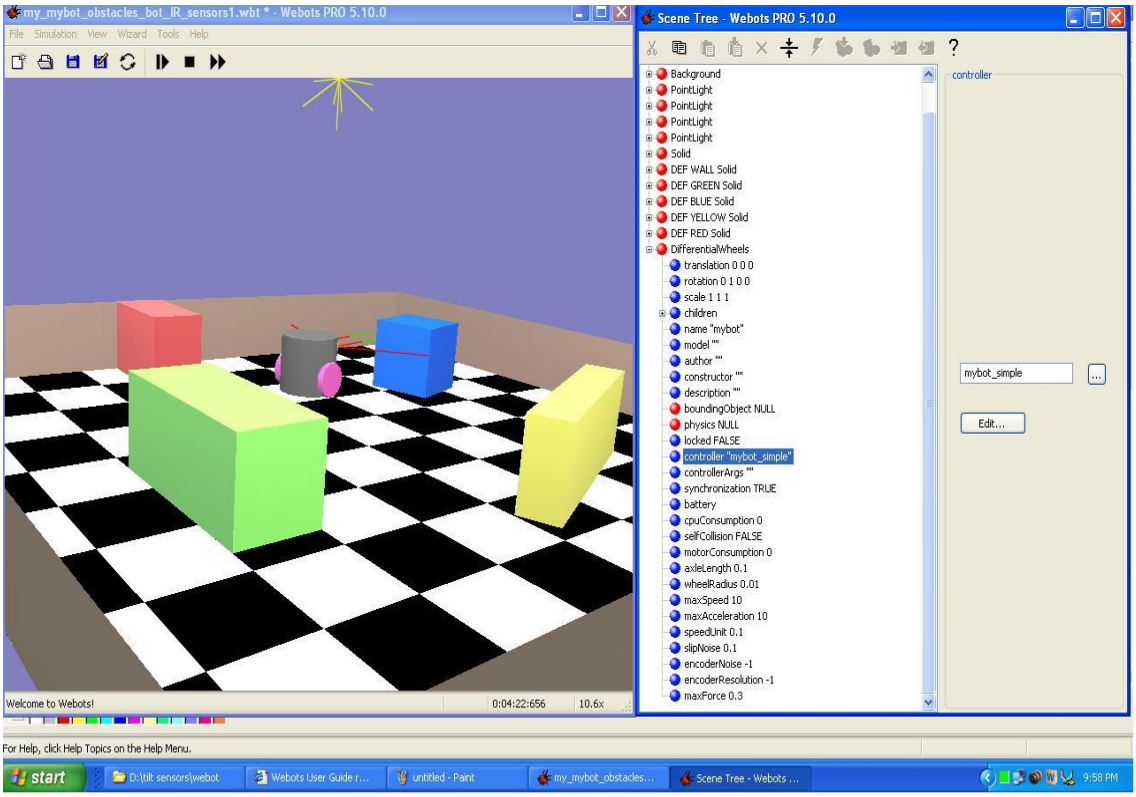


Figure 4.4: My Bot avoiding Third Obstacle (Blue)

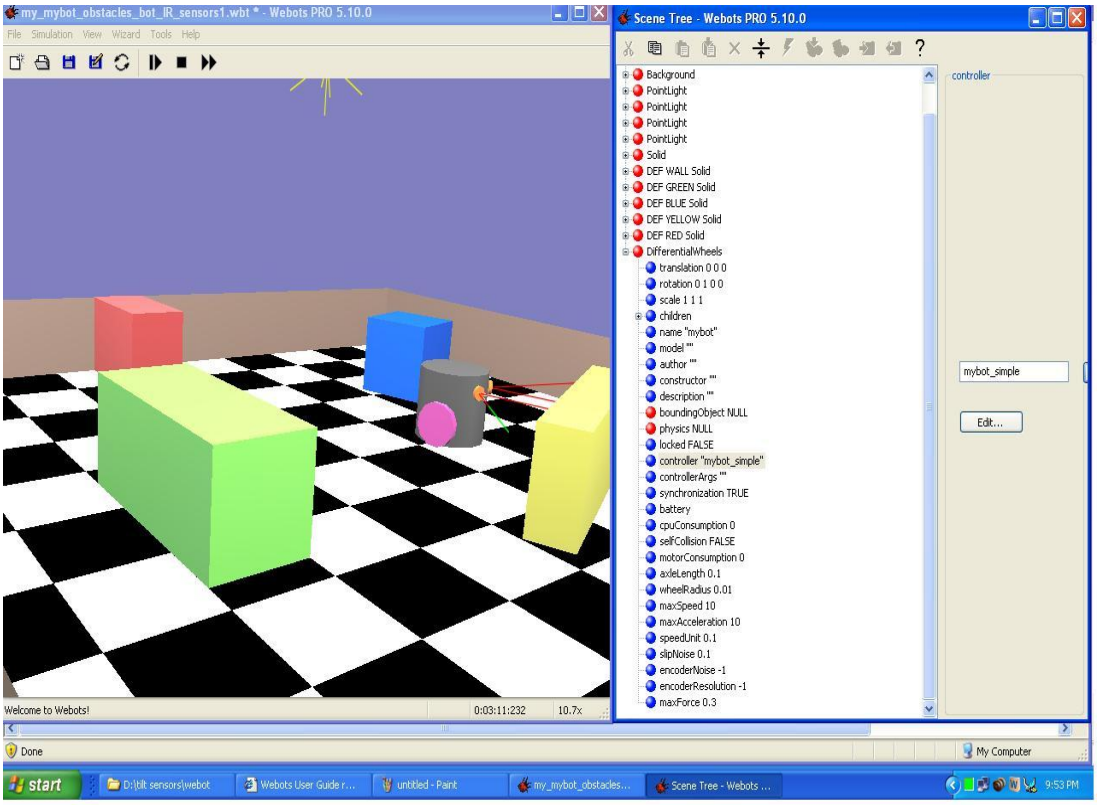


Figure 4.5: My Bot avoiding Fourth Obstacle (Yellow)

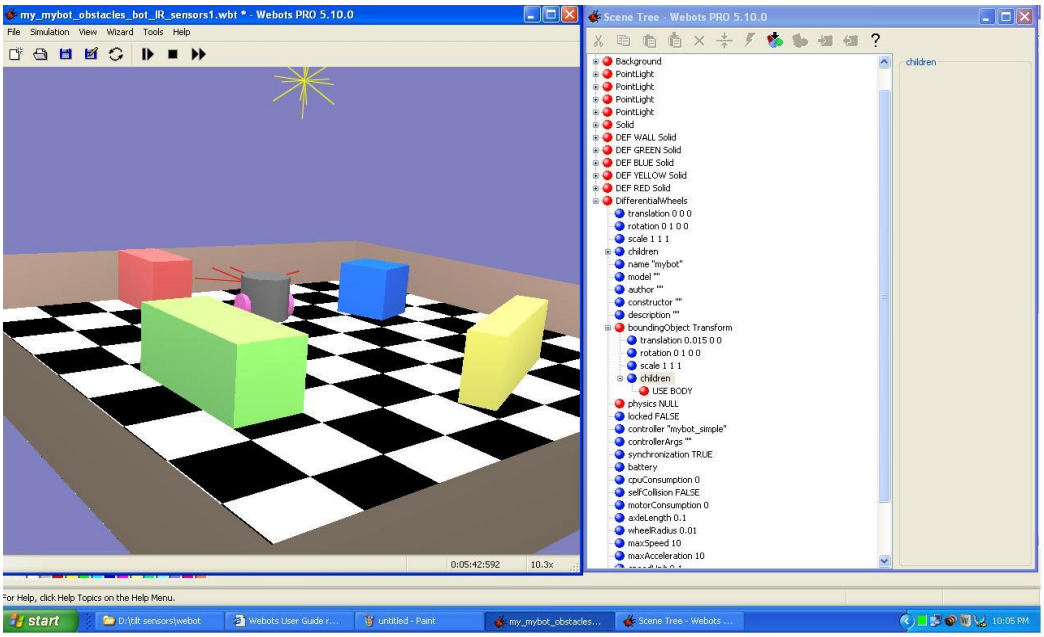
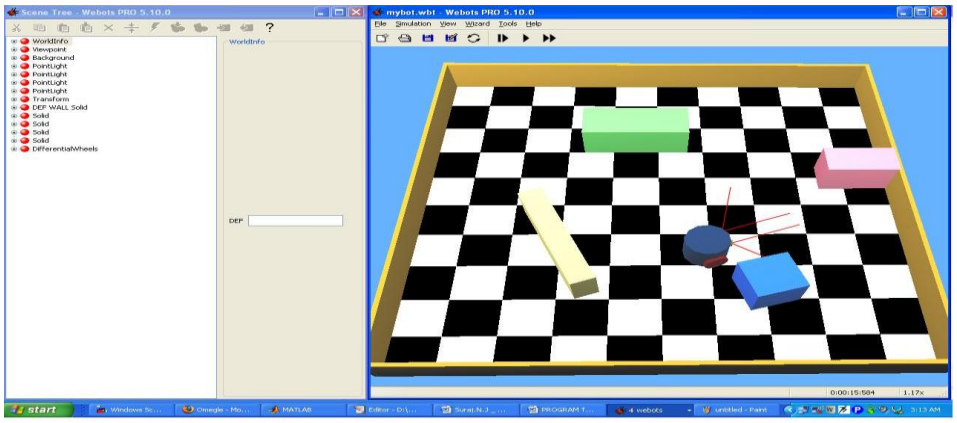


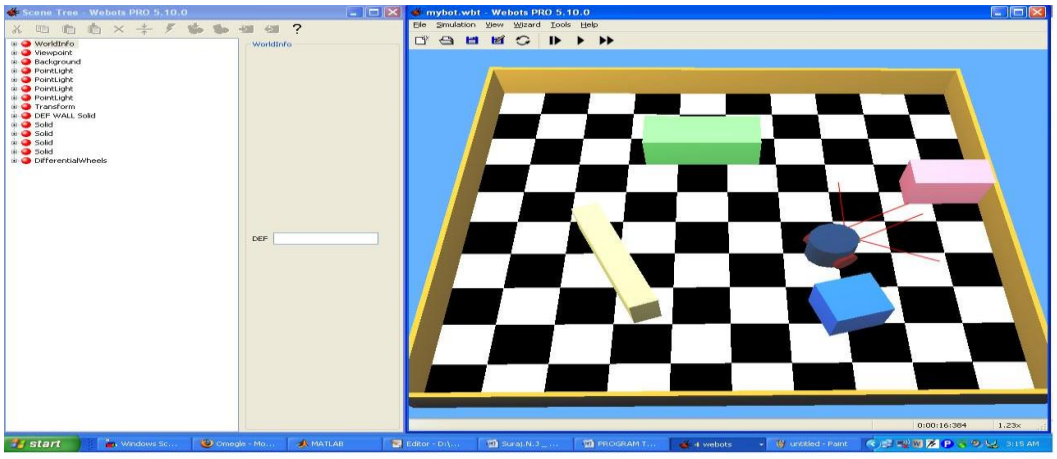
Figure 4.6: My Bot being Bounded and locked in its Final Position, preventing its further motion.

PATH-TRAVELLED BY BOT AFTER FUZZY IMPLEMENTATION

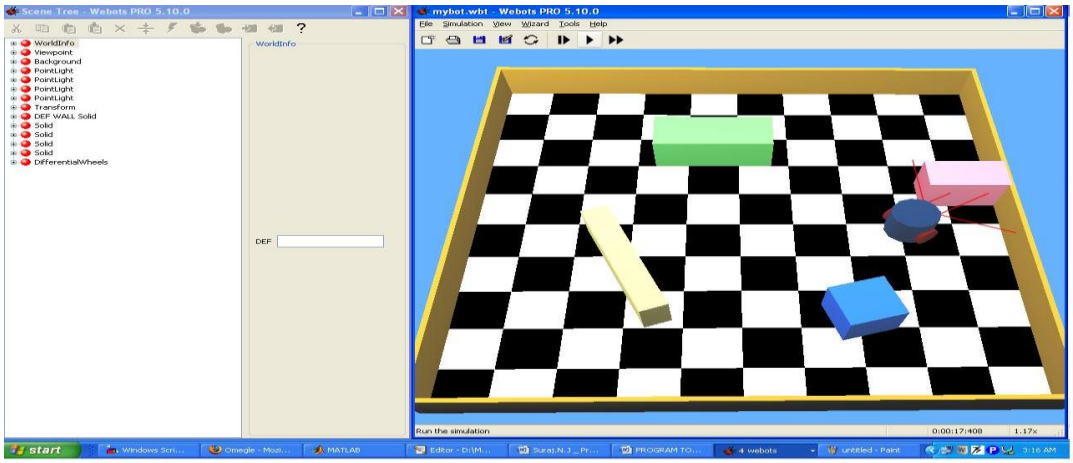
4.7.



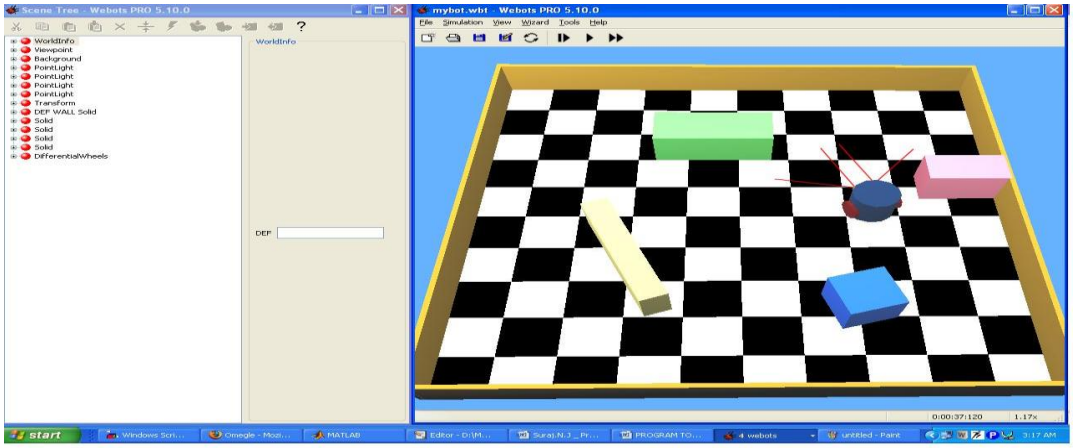
4.8.



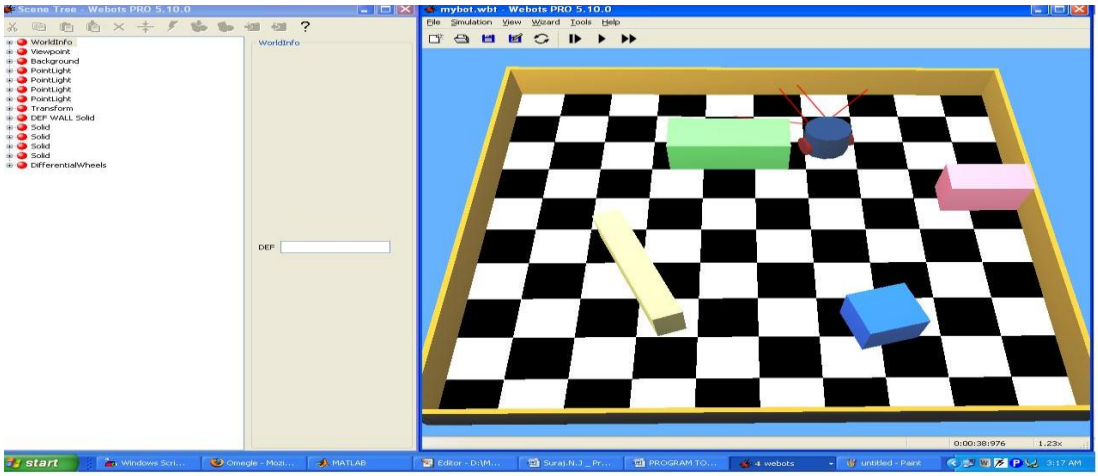
4.9.


Scene Tree - Webots PRO 5.10.0
WorldInfo
Viewpoint
Background
PointLight
PointLight
PointLight
PointLight
Transform
DEF WALL Solid
Solid
Solid
Solid
Solid
DifferentialWheels
DEF
mybot.wbt - Webots PRO 5.10.0
File Simulation View Wizard Tools Help
Run the simulation
0:00:17:408
1.17x
start
3:16 AM

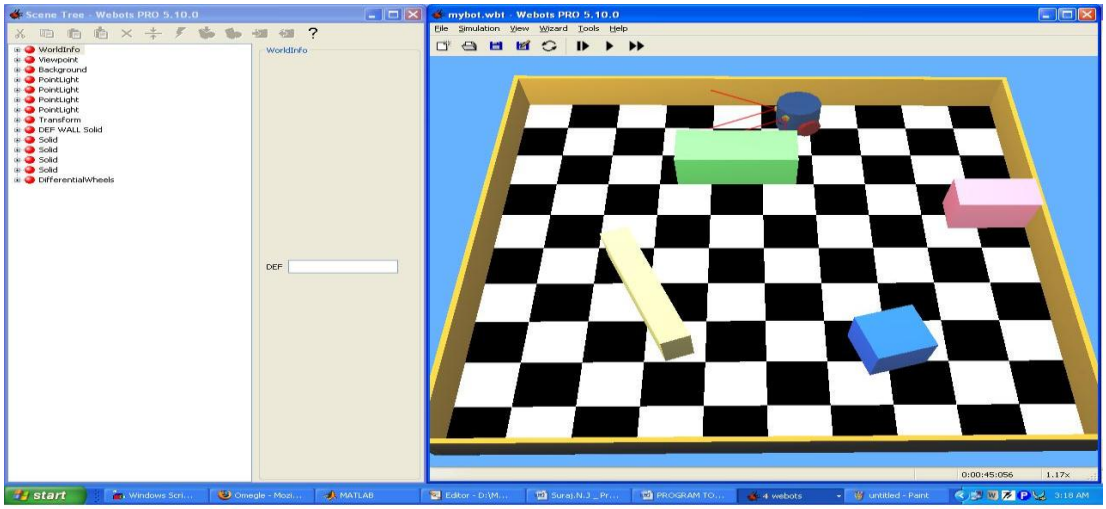
4.10.


Scene Tree - Webots PRO 5.10.0
WorldInfo
Viewpoint
Background
PointLight
PointLight
PointLight
PointLight
Transform
DEF WALL Solid
Solid
Solid
Solid
Solid
DifferentialWheels
DEF
mybot.wbt - Webots PRO 5.10.0
File Simulation View Wizard Tools Help
0:00:37:120
1.17x
start
3:17 AM

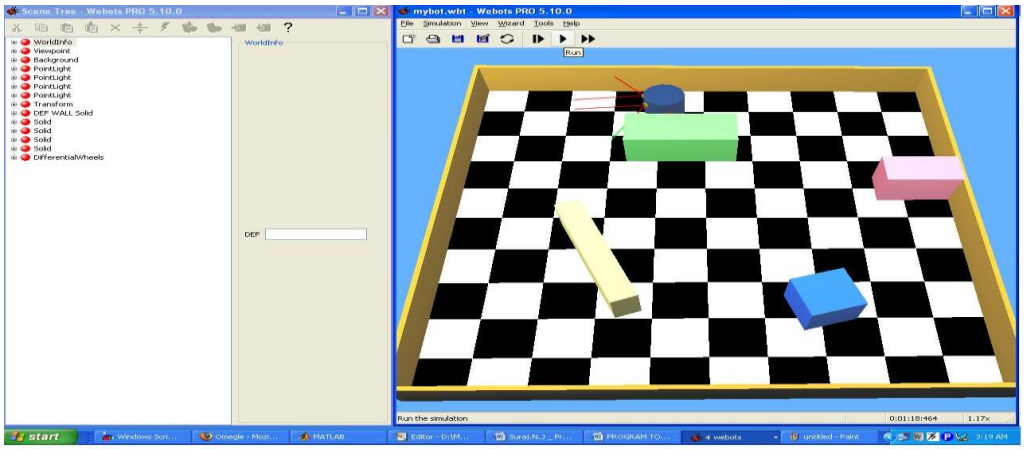
4.11.


Scene Tree - Webots PRO 5.10.0
WorldInfo
Viewpoint
Background
PointLight
PointLight
PointLight
PointLight
Transform
DEF WALL Solid
Solid
Solid
Solid
Solid
DifferentialWheels
DEF
mybot.wbt - Webots PRO 5.10.0
File Simulation View Wizard Tools Help
0:00:38:976
1.23x
start
Windows Scri...
Omegle - Mozi...
MATLAB
Editor - D:\M...
Suraj.N.J _ Pr...
PROGRAM TO...
4 webots
untitled - Paint
3:17 AM

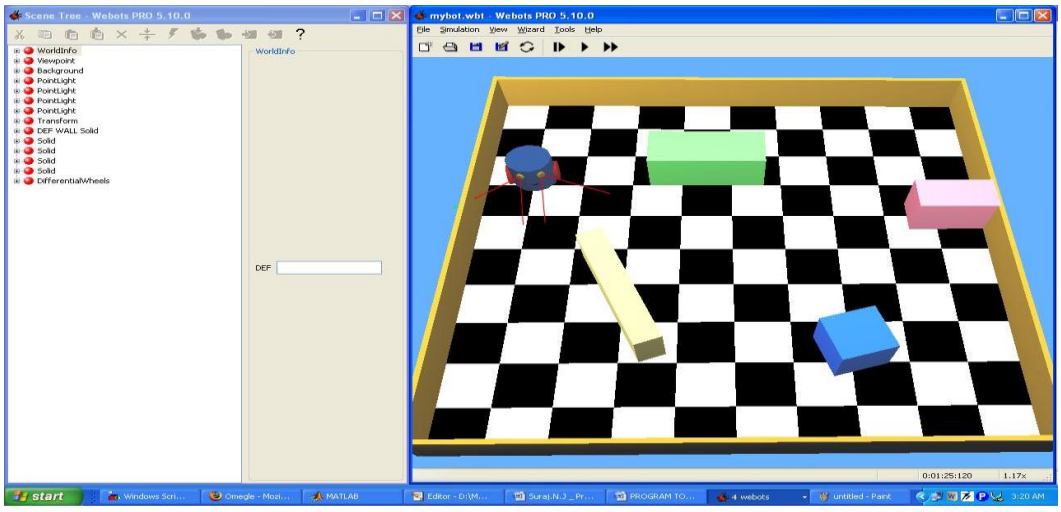
4.12.


Scene Tree - Webots PRO 5.10.0
WorldInfo
Viewpoint
Background
PointLight
PointLight
PointLight
PointLight
Transform
DEF WALL Solid
Solid
Solid
Solid
Solid
DifferentialWheels
DEF
mybot.wbt - Webots PRO 5.10.0
File Simulation View Wizard Tools Help
0:00:45:056
1.17x
start
Windows Scri...
Omegle - Mozi...
MATLAB
Editor - D:\M...
Suraj.N.J _ Pr...
PROGRAM TO...
4 webots
untitled - Paint
3:18 AM

4.13.


Scene Tree - Webots PRO 5.10.0
WorldInfo
Viewpoint
Background
PointLight
PointLight
PointLight
PointLight
Transform
DEF WALL Solid
Solid
Solid
Solid
Solid
DifferentialWheels
DEF
mybot.wbt - Webots PRO 5.10.0
File Simulation View Wizard Tools Help
Run
Run the simulation
0:01:18:464
1.17x
start

4.14.


Scene Tree - Webots PRO 5.10.0
WorldInfo
Viewpoint
Background
PointLight
PointLight
PointLight
PointLight
Transform
DEF WALL Solid
Solid
Solid
Solid
Solid
DifferentialWheels
DEF
mybot.wbt - Webots PRO 5.10.0
File Simulation View Wizard Tools Help
0:01:25:120
1.17x
start

Chapter 5

CONCLUSION

CONCLUSION

The program written in C for the controller simply reads the sensor values and sets the two motors' speed in such a way that MyBot avoids the obstacles.

Suitable maximum speed, maximum acceleration, size of wheels can all be set in WEBOT SIMULATOR.

The position of **MyBot** can be contained by putting a bounding object, and it can be constrained to avoid obstacles only along the required directions, and continue its motion along the other ones.

Fuzzy-Logic is widely used in Automobile and other vehicle sub-systems, such as ABS, ACC, etc. It is also used extensively in Air-Conditioners, Cameras, Digital Image Processing (Edge Detection), Pattern Recognition in Remote Sensing, Elevators, Rice Cookers, and many more.

Fuzzy-Logic and WEBOTS provide robustness and adaptively features with respect to environmental or structural changes of the ROBOT.

The solutions found by human-reasoning (Fuzzy-Logic) are simple and in many cases they work in different environmental situations. The obtained results suggest that Fuzzy-Logic is a suitable robust tool for synthesizing controllers for a number of robots and well-simulated and analysed using WEBOTS Simulator.

One can easily find that, PATH travelled by BOT is better and optimized after Fuzzy-Implementation and BOT is consciously taking its decisions to achieve the goal and avoid obstacles intelligently, adjusting its wheel speed.

Chapter 6

DISCUSSION

DISCUSSION

As discussed before, the strategy adopted for Fuzzy implementation was to get the values (Distance-sensor) values from Webots and include into Fuzzy-logic Code and plot it in Mamdani Matlab controller FIS Editor, and this gives not too accurate results.

The suggestion to improve upon the result, is to include the Fuzzy-Logic Code into Webots controller Code and increase the complexity of the Fuzzy-Logic Code.

In this way, the entire operation will be automatically calculated and recalculated to give the best output.

One more efficient way to increase the performance of this WEBOTS simulation is to go for Neuro-Fuzzy Implementation.

Chapter 7

REFERENCES

REFERENCES

1. K.Liu & F.L.Lewis, Fuzzy Logic based Navigation Controller For An Autonomous Mobile Robot, IEEE, Vol. 7, pp.49-63,1989.
2. Dan Simon & Vamsi Mohan Peri, Fuzzy Logic Control For An Autonomous Robot, IEEE Competition, Vol. 6, No.4, pp.445-454, 1990.
3. S.K.Harisha, Ramkanth Kumar P., S.C.Sharma, Fuzzy Logic Reasoning To Control Mobile Robot On Pre-Defined Strip Path, Proceedings of World Academy of Science, Engineering and Technology, Vol. 4, pp.539-546,2003.
4. Oscar Castillo, Luis T. Aguilar, Selene Cardinas, Fuzzy Logic Tracking Control For Unicycle Mobile Robots, Engineering Letters, pp. 73-82, 2003.
5. Anis Fatmi, Amur Al Yahmadi, Lazhar Khriji, Nouri Masmoudi, A Fuzzy Logic Based Navigation of a Mobile Robot, Proceedings of World Academy of Science, Engineering and Technology, pp. 159-177, 1995.
6. Rajiv Malhotra, Atri Sarkar, Development of a Fuzzy Logic Based Mobile Robot for Dynamic Obstacle Avoidance and Goal Acquistion in an Unstructured Environment, Proceedings of IEEE/ASME, International Conference on Advanced Intelligent Mechatronics, pp.235-247, 2003.
7. Harmeet Singh, Sanchit Arora, Using Fuzzy Logic for Mobile Robot Control, IIT-D manuals, India, 2006.
8. Nelson Ramos, Pedro U.Lima, Joao M.C. Sousa, Robot Behaviour Coordination Based On Fuzzy Decision-Making, GCAR, IDMEC, Portugal, pp. 145-172.
9. D.Busquets, C.Sierra, R.Lopez De Mantaras, A Multi-Agent Approach to Fuzzy Landmark-Based Navigation, Multi-Valued Logic & Soft Computing, pp.971-978, 1995.
10. K.Figueiredo, M.Vellasco, M.A. Pacheco, Mobile robot control using intelligent agents, Second international workshop on Intelligent systems design and application, pp.159-173, 1999.
11. Olivier Michel, Webots: Symbiosis between Virtual and Real Mobile Robots, Springer Berlin / Heidelberg, pp. 306-318, 1998.
12. Li Wei, Chen Zushun, Ma Chenyu, Fuzzy Logic based Behaviour Fusion for Navigation of an Intelligent Mobile Robot, Journal of Computer Science & Technology, pp. 125-132, 1999.
13. WEBOTS teaching material, Cyberbotics Corporation.
14. Jan Jantzen, Tutorial on Fuzzy-Logic, TU Denmark, Automation, 2008.
15. MATLAB tutorials.