ECE 230 DP04 Final Report

Xiangbo Cai

1. Introduction

Pipeline inspection in urban environments presents persistent challenges. Pipes, intricate in their construction, are often buried beneath the ground, making detection a formidable task. Resorting to surface excavation for inspection not only risks damaging existing infrastructure but also contributes to soil erosion. To address these issues, non-destructive evaluation (NDE) technology has emerged as a solution. NDE utilizes principles such as radiation, ultrasonics [1], infrared [2], electromagnetic fields, and eddy current [3] to assess materials, parts, and equipment without compromising their future functionality or current operational status.

In this report, the groundbreaking solution is introduced like this: a pipeline inspection robot [4] designed to supplant human efforts and obviate the need for disruptive soil excavation during underground inspections. Compact and agile, the robot seamlessly navigates pipelines, collecting crucial parameters and information. It evaluates the condition of the pipeline, promptly notifying of any repair needs. Moreover, it autonomously executes maintenance tasks underground, effectively substituting for human intervention. This innovative pipeline robot epitomizes the essence of non-destructive testing.

The primary objective of this article is to elucidate the technical underpinnings of pipeline non-destructive inspection robots using Verilog. Input parameters that the robot may require are meticulously constructed, alongside their corresponding output parameters. Leveraging Vivado Verilog code, the functionalities of the pipeline robot are implemented. Furthermore, two comprehensive testbench are also furnished, employing generated waveforms to simulate and address real-world challenges. Through the research on pipeline non-destructive evaluation robots, this robot aims to furnish indispensable technical support and assurance for the secure operation and sustainable development of urban pipelines.

2. Robot Parameters

The pipe maintenance robot is equipped with essential features to navigate and perform maintenance tasks within pipelines. Those parameters can precisely control the robot and help achieve the result expected.

Pipe Maintenance Robot's Finite State Machine (FSM) has featured the following inputs. A feature with [On, Off] button for control and a sensor with an 8-bit register to store its starting location, represented as [4b_x, 4b_y]. Additionally, it possesses a sensor to detect maintenance requirements in each physical space, indicating tasks such as Fire, Cool, Plunge, or Free. A compass aid in orientating the robot, with directional indications of North, East, South, or West

[N, E, S, W]. Furthermore, it employs sensors to detect wall locations relative to its forward-facing direction, distinguishing between Left, Right, and Forward [L, R, F]. Crucially, the robot maintains a register to store its current location, facing direction, and states, facilitating efficient navigation and task execution within the pipeline environment. All sensor inputs are active-high and represented as one-hot vectors, except for the location vector, ensuring precise control and effective operation of the robot.

The outputs of the pipe maintenance robot's FSM are essential commands and information necessary for its operation and navigation within the pipeline. It includes instructions for turning, denoted as Turn Left or Turn Right, to adjust its orientation as needed. Additionally, there is an output to drive forward, enabling the robot to progress through the pipeline efficiently. The current location output, represented as [4b_x, 4b_y], provides crucial positional information for tracking the robot's movement along the pipeline. Moreover, an output indicating the action to be performed, such as Fire, Cool, or Plunge, guides the robot in executing maintenance tasks as dictated by the pipeline's condition. All outputs are active-high and conveyed as one-hot vectors, ensuring precise control and effective communication between the robot and its operational environment.

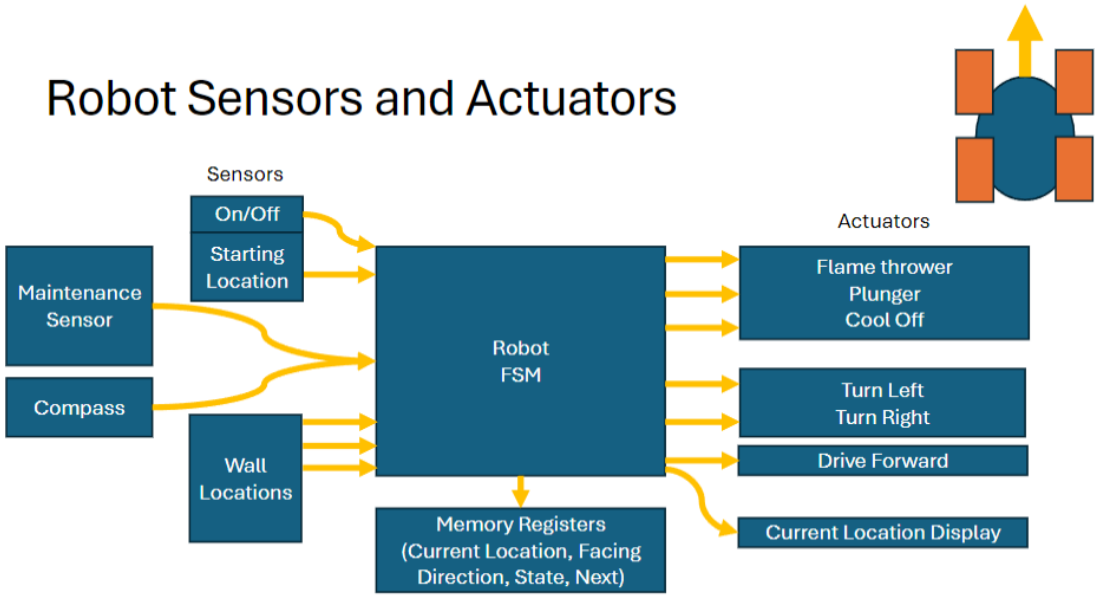

Figure 1: The Input and Output parameters of the robot FSM

According to the input and output information above, the related FSM diagram can be drawn as follows (with related inputs and outputs aside):

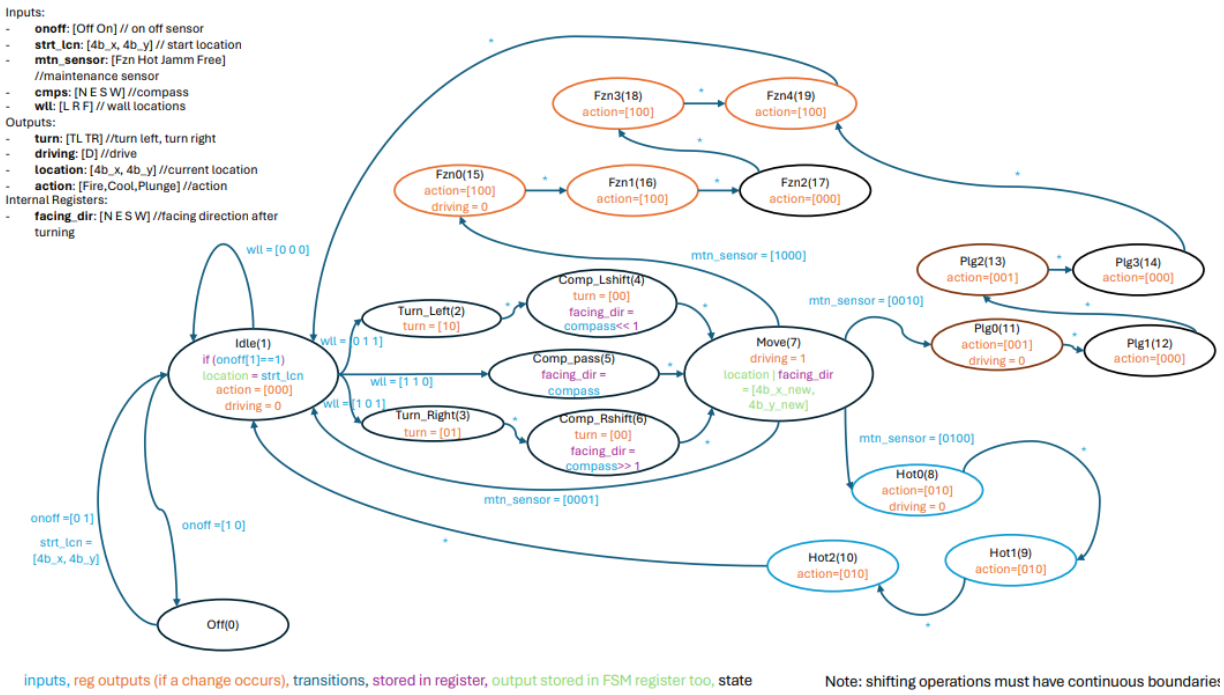

Figure 2: The Finite State Machine (FSM) of the pipe maintenance robot

3. Simulating in Vivado Verilog and Related Testbench

According to the FSM state machine description drawn above, Verilog code based on the corresponding input and output can be built, and the corresponding functions can also be implemented. Corresponding code references are provided in the appendix. This code implements the state transition logic and action execution of the pipeline maintenance robot and ensures that it can accurately control the robot's behavior based on external input.

In addition, in order to better demonstrate the research results, the corresponding simulation data is provided. As shown in Figures 3 and 4, this article provides simulation routes of two different robots and their corresponding data. These data not only verify the effectiveness of the model, but also demonstrate its application potential in different scenarios. Relevant test code and more detailed data can be found in the appendix to facilitate readers to deeply understand and reproduce the research results.

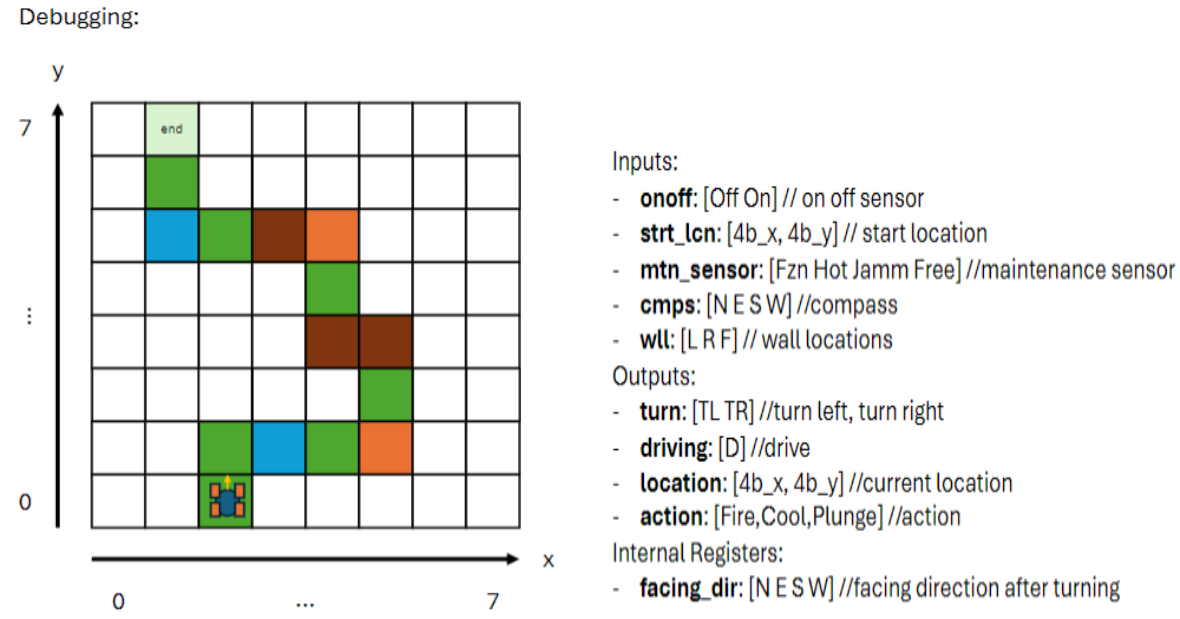

Figure 3: The first simulation test process (diagram), testbench code can be seen in appendix.

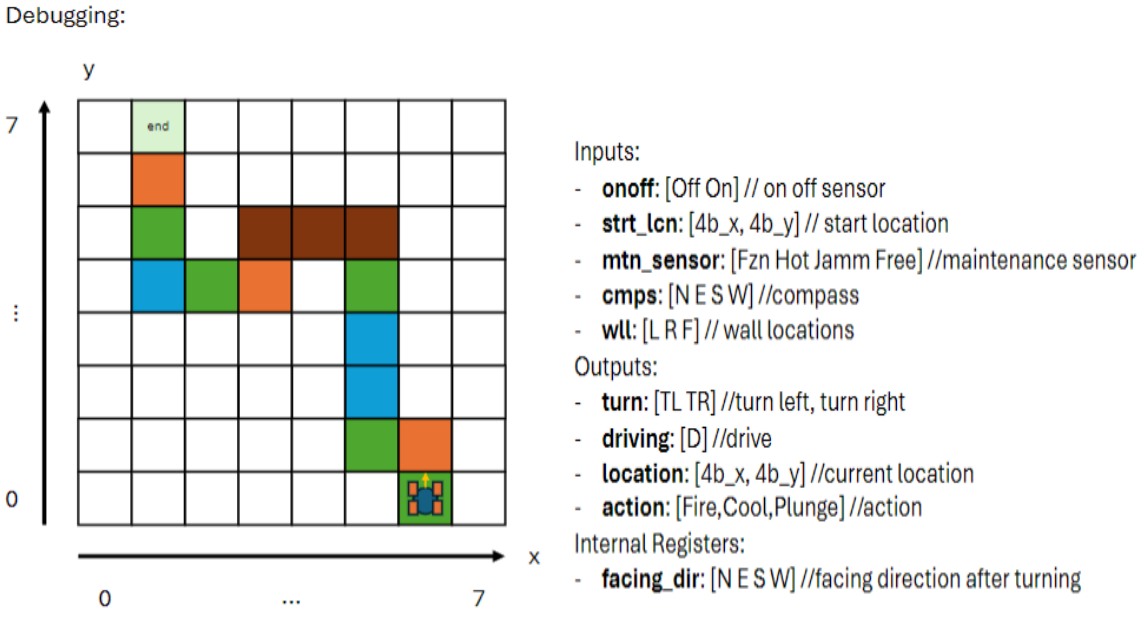

Figure 4: The second simulation test process (diagram), testbench code can be seen in appendix.

4. The Simulation Result - Waveforms

In this section, it will be shown that the waveform plots of two different simulation results create representative models for this report. These waveforms are obtained by executing Verilog code in the Vivado environment, specifically after running the corresponding testbench. These waveform diagrams not only visually display the simulation results, but also present the dynamic change process of the data, reflecting the system's response and performance under different test conditions. Through these detailed and informative waveform diagrams, readers can more clearly understand and analyze the effectiveness of the design and its potential application value.

Here is the Figure 3 test process (test one) result, figure5 will showcase an overall simulation results and Figure 6,7,8 will show the frozen, hot and plunge circle generate results.

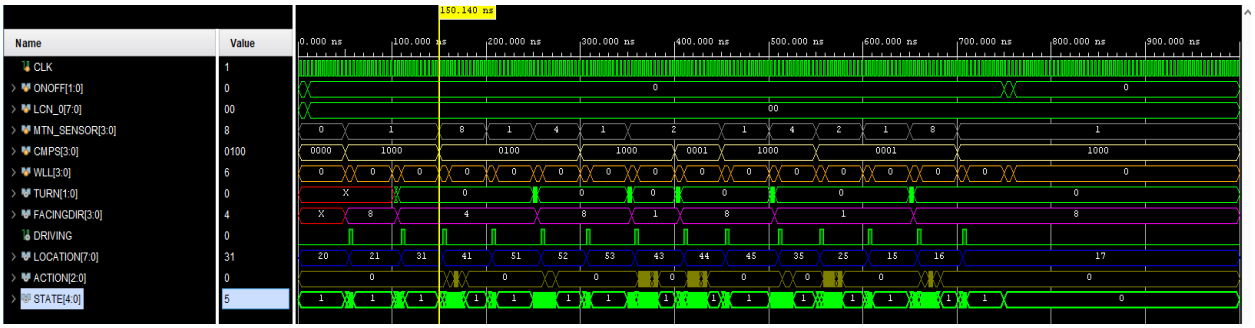

Figure 5: An overview of the test one whole simulation result

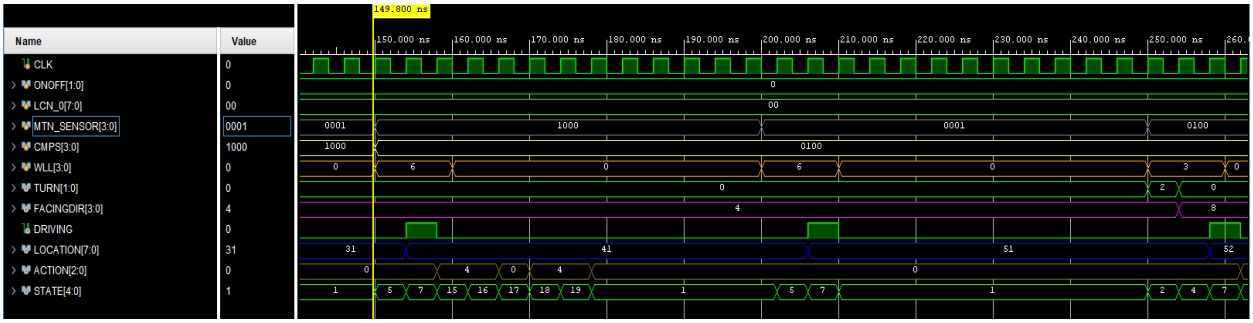

Figure 6: When the mtn_sensor=1000, frozen mode executed, iterate 15,16,17,18,19 states.

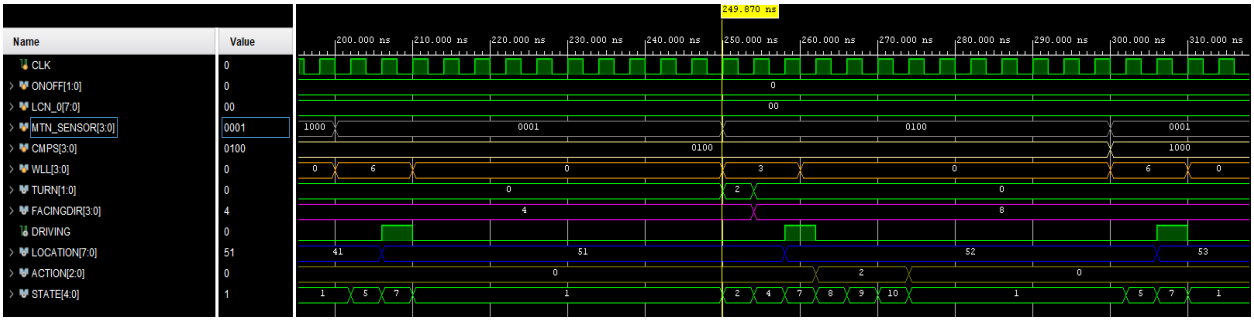

Figure 7: When the mtn_sensor = 0100, hot mode executed, iterate 8,9,10 states.

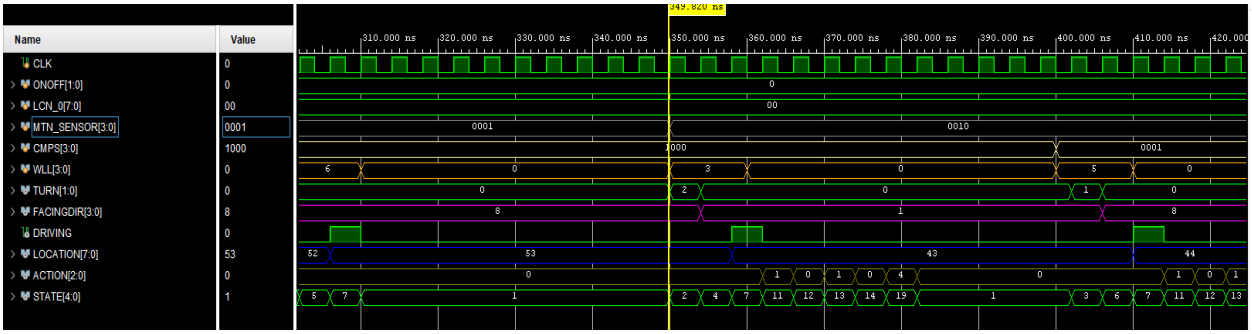

Figure 8: When the mtn_sensor=0010, plunge mode executed, iterate 11,12,13,14,19 states.

And the follow is Figure 4's test (test two) process executed result, the Figure 9 is and overall simulation result and Figure 10, 11, 12 will be the hot, frozen and plunge simulation circle results.

Figure 9: An overview of test two whole simulation result

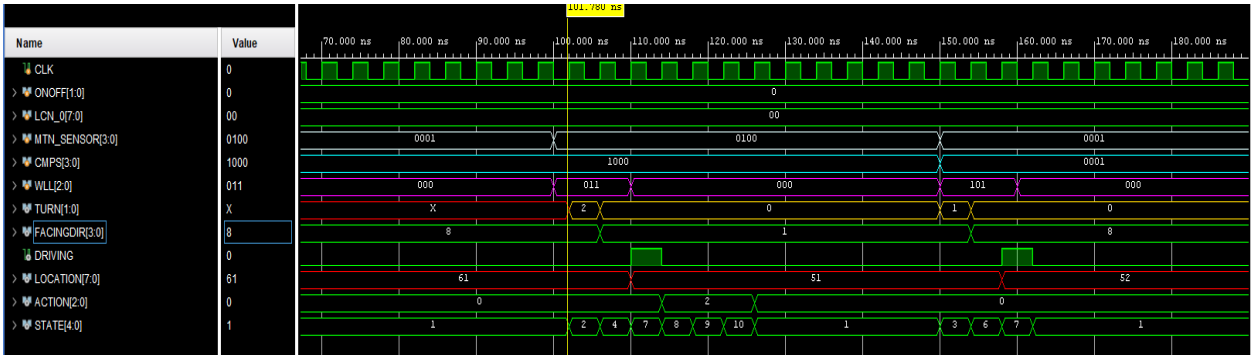

Figure 10: When the mtn_sensor = 0100, hot mode executed, iterate 8,9,10 states.

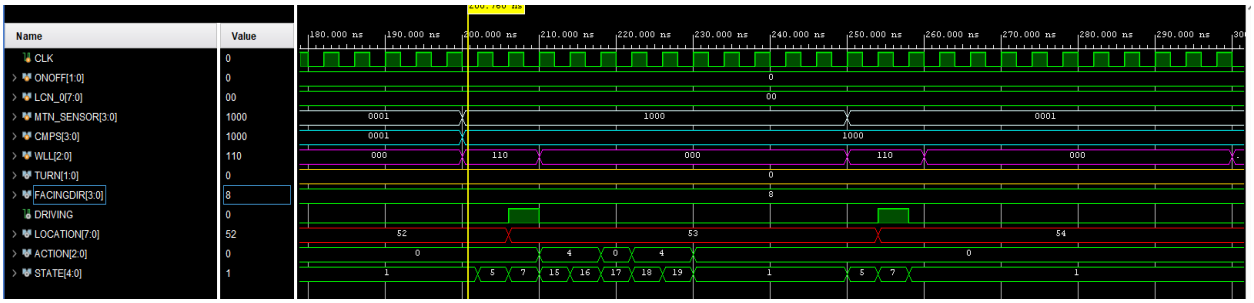

Figure 11: When the mtn_sensor = 1000, frozen mode executed, iterate 15,16,17,18,19 states.

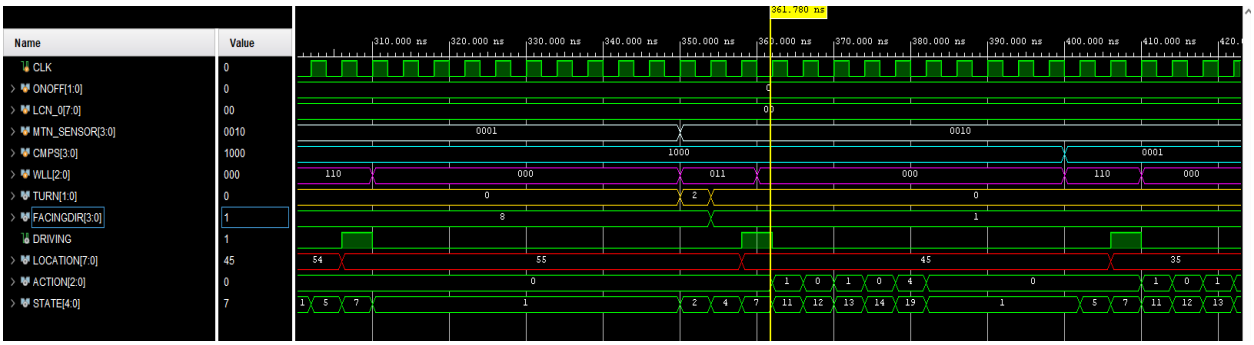

Figure 12: When the mtn_sensor = 0010, plunge mode executed, iterate 11,12,13,14,19 states.

5. Conclusion

As society continues to demand more advanced ideas, there will be more and more engineering challenges encountered. The importance of NDE technology becomes increasingly prominent—and the need for developing more technology like this becoming ever-necessary. The pipeline maintenance robot introduced in this article is a typical example of NDE technology in real-life applications. The robot utilizes advanced non-destructive testing methods, like the robot concept, and implements related functions, which has significantly promoted the inspection and maintenance of urban pipelines, through Verilog programming.

Furthermore, the design of a pipeline robot first requires precise design of the result graph of the finite state machine (FSM), and then writing and executing corresponding code based on these design results. During operation, the robot can identify the direction, detect various conditions within the pipeline, such as whether the temperature is abnormal (overheating or undercooling), and generate corresponding feedback through internal programs based on the detection results. In addition, the robot can detect based on the position information on the left, right and front, and determine the coordinate position of its next move based on these detection data.

Notably, there is still room for improvement in the design of pipeline robots. When inputting temperature abnormality information and judging direction parameters, the robot currently only sets 0 or 1 to indicate its status. This simplified processing method cannot meet the needs of more complex practical applications. It is expected that in future development, with the advancement of technology and optimization of parameter settings, that these problems will be rendered solved--making the functions of pipeline inspection robots' truly accurate and complete.

6. References

[1] J. H. Kurz, A. Jüngert, S. Dugan, G. Dobmann, and C. Boller, "Reliability considerations of NDT by probability of detection (POD) determination using ultrasound phased array," *Engineering Failure Analysis*, vol. 35, pp. 609–617, Dec. 2013. doi:10.1016/j.engfailanal.2013.06.008

[2] C. Ibarra-Castanedo *et al.*, "Infrared vision for artwork and Cultural Heritage Nde Studies: Principles and case studies," *Insight - Non-Destructive Testing and Condition Monitoring*, vol. 59, no. 5, pp. 243–248, May 2017. doi:10.1784/insi.2017.59.5.243

[3] Guang Yang *et al.*, "Pulsed eddy-current based giant magnetoresistive system for the inspection of Aircraft Structures," *IEEE Transactions on Magnetics*, vol. 46, no. 3, pp. 910–917, Mar. 2010. doi:10.1109/tmag.2009.2032330

[4] Y.-S. Kwon and B.-J. Yi, "Design and motion planning of a two-module collaborative indoor pipeline inspection robot," *IEEE Transactions on Robotics*, vol. 28, no. 3, pp. 681–696, Jun. 2012. doi:10.1109/tro.2012.2183049

7. Addendum | Associated Code for the Waveform

Robot Module Design Verilog:

```
module pipeFSM(
    input clk,
    input [1:0] onoff, // [off,on]
    input [7:0] strt_lcn, // Assuming 4-bit for each x and y
    input [3:0] mtn_sensor, // Encoding Fzn, Hot, Jamm, Free
    input [3:0] cmps,  // Compass direction  [N,E,S,W]
    input [2:0] wll, // Wall location [L, R, F]

    output reg [1:0] turn, // Turn Left, Turn Right
    output reg driving, // Drive
    output reg [7:0] location, // Current location
    output reg [2:0] action,// Fire, Cool, Plunge
    output reg [3:0] facing_dir, // facing turn, [N,E,S,W]

    output reg [4:0] State
    );

    parameter Off       = 0;
    parameter Idle      = 1;
```

```
parameter Turn_Left = 2;
parameter Turn_Right= 3;
parameter Comp_Lshift=4;
parameter Comp_pass= 5;
parameter Comp_Rshift=6;
parameter Move      = 7;
parameter Hot0      = 8;
parameter Hot1      = 9;
parameter Hot2      = 10;
parameter Plg0      = 11;
parameter Plg1      = 12;
parameter Plg2      = 13;
parameter Plg3      = 14;
parameter Fzn0      = 15;
parameter Fzn1      = 16;
parameter Fzn2      = 17;
parameter Fzn3      = 18;
parameter Fzn4      = 19;



  initial begin State = Idle; end

  always @(posedge clk) begin

          // State transitions
       case (State)

             Off: if ((onoff == 2'b10) && (strt_lcn != 0))  State = Idle;

             Idle:  begin
                   if (onoff == 2'b10) State = Off;
                   else if (wll == 3'b000) State = Idle;
                   else if (wll == 3'b011) State = Turn_Left;
                   else if (wll == 3'b110) State = Comp_pass;
                   else if (wll == 3'b101) State = Turn_Right;
              end



              Turn_Left: State = Comp_Lshift;
              Turn_Right: State = Comp_Rshift;
              Comp_Lshift: State = Move;
              Comp_pass: State = Move;
              Comp_Rshift: State = Move;

              Move: begin
                    if (mtn_sensor == 4'b1000) State = Fzn0;
                    else if (mtn_sensor == 4'b0010) State = Plg0;
                    else if (mtn_sensor == 4'b0100) State = Hot0;
                    else if (mtn_sensor == 4'b0001) State = Idle;
              end

              Hot0: State = Hot1;
              Hot1: State = Hot2;
              Hot2: State = Idle;

              Plg0: State = Plg1;
              Plg1: State = Plg2;
              Plg2: State = Plg3;
```

```
            Plg3: State = Fzn4;

            Fzn0: State = Fzn1;
            Fzn1: State = Fzn2;
            Fzn2: State = Fzn3;
            Fzn3: State = Fzn4;
            Fzn4: State = Idle;

        endcase
end

always @(State) begin
   // State actions
   case (State)
      Idle:begin
      if (onoff == 2'b01) begin
           location = strt_lcn; end
           action = 3'b000;
           driving = 0;
      end
       Turn_Left: turn = 2'b10;
       Turn_Right: turn = 2'b01;

       Comp_Lshift:begin
           turn = 2'b00;
           facing_dir = (cmps << 1) + cmps[3];
       end

       Comp_pass: facing_dir = cmps;

       Comp_Rshift: begin
            turn = 2'b00;
            facing_dir = (cmps >> 1) +(cmps[0] << 3);
       end

       Move: begin
          driving = 1;
          if (facing_dir == 4'b0001) begin
              location = location - 16;
          end else if (facing_dir == 4'b0010) begin
              location = location - 1;
          end else if (facing_dir == 4'b0100) begin
              location = location + 16;
          end else if (facing_dir == 4'b1000) begin
              location = location + 1;
          end
       end


       Hot0:begin
           action = 3'b010;
           driving = 0;
       end

       Hot1: action = 3'b010;
       Hot2: action = 3'b010;

       Plg0:begin
           action = 3'b001;
           driving = 0;
       end
```

```
            Plg1: action = 3'b000;
            Plg2: action = 3'b001;
            Plg3: action = 3'b000;

            Fzn0:begin
                action = 3'b100;
                driving = 0;
            end

            Fzn1: action = 3'b100;
            Fzn2: action = 3'b000;
            Fzn3: action = 3'b100;
            Fzn4: action = 3'b100;

        endcase
    end

endmodule
```

Test 1 testbench code:

```
module pipeFSM_1;

    reg CLK; //clock
    reg [1:0] ONOFF; // on/off
    reg [7:0] LCN_0; //starting location
    reg [3:0] MTN_SENSOR; //maintenance sensor
    reg [3:0] CMPS; //compass
    reg [3:0] WLL; //wall locations

    wire [1:0] TURN;
    wire [3:0] FACINGDIR;
    wire DRIVING;
    wire [7:0] LOCATION;
    wire [2:0] ACTION;
    wire [4:0] STATE;

    // Clock generation
    initial begin
        CLK = 0;
        forever #2 CLK = ~CLK;
    end

    pipeFSM
inst0(CLK,ONOFF,LCN_0,MTN_SENSOR,CMPS,WLL,TURN,FACINGDIR,DRIVING,LOCATION,ACTION,STATE);

    initial begin
    ONOFF=2'b01; LCN_0=8'b00100000; MTN_SENSOR=4'b0000; CMPS=4'b0000; WLL=3'b000;
    #10 ONOFF=2'b00; LCN_0=8'b00000000; MTN_SENSOR=4'b0000; CMPS=4'b0000; WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b101;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b1000; CMPS=4'b0100; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b0100; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0100; CMPS=4'b0100; WLL=3'b011;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0010; CMPS=4'b1000; WLL=3'b011;
```

```
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0010; CMPS=4'b0001; WLL=3'b101;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0100; CMPS=4'b1000; WLL=3'b011;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0010; CMPS=4'b0001; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b0001; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b1000; CMPS=4'b0001; WLL=3'b101;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;
    #40 ONOFF=2'b10; MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 ONOFF=2'b00; WLL=3'b000;
    end


endmodule
```

Test 2 testbench code:

```
module pipeFSM_tb_2;

    reg CLK; //clock
    reg [1:0] ONOFF; // on/off
    reg [7:0] LCN_0; //starting location
    reg [3:0] MTN_SENSOR; //maintenance sensor
    reg [3:0] CMPS; //compass
    reg [3:0] WLL; //wall locations

    wire [1:0] TURN;
    wire [3:0] FACINGDIR;
    wire DRIVING;
    wire [7:0] LOCATION;
    wire [2:0] ACTION;
    wire [4:0] STATE;

    // Clock generation
    initial begin
        CLK = 0;
        forever #2 CLK = ~CLK;
    end

    pipeFSM
inst0(CLK,ONOFF,LCN_0,MTN_SENSOR,CMPS,WLL,TURN,DRIVING,LOCATION,ACTION,FACINGDIR,STATE);

    initial begin
    ONOFF=2'b01; LCN_0=8'b01100000; MTN_SENSOR=4'b0000; CMPS=4'b0000; WLL=3'b000;

//Initiate
    #10 ONOFF=2'b00; LCN_0=8'b00000000; MTN_SENSOR=4'b0000; CMPS=4'b0000; WLL=3'b000;
//block 1
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;

//block2
    #40 MTN_SENSOR=4'b0100; CMPS=4'b1000; WLL=3'b011;
    #10 WLL=3'b000;
```

```
//block3
    #40 MTN_SENSOR=4'b0001; CMPS=4'b0001; WLL=3'b101;
    #10 WLL=3'b000;

//block4
    #40 MTN_SENSOR=4'b1000; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;

//block5
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;

//block6
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;

//block7
    #40 MTN_SENSOR=4'b0010; CMPS=4'b1000; WLL=3'b011;
    #10 WLL=3'b000;

//block8
    #40 MTN_SENSOR=4'b0010; CMPS=4'b0001; WLL=3'b110;
    #10 WLL=3'b000;

//block9
    #40 MTN_SENSOR=4'b0010; CMPS=4'b0001; WLL=3'b011;
    #10 WLL=3'b000;

//block10
    #40 MTN_SENSOR=4'b0100; CMPS=4'b0010; WLL=3'b101;
    #10 WLL=3'b000;

//block11
    #40 MTN_SENSOR=4'b0001; CMPS=4'b0001; WLL=3'b110;
    #10 WLL=3'b000;

//block12
    #40 MTN_SENSOR=4'b1000; CMPS=4'b0001; WLL=3'b101;
    #10 WLL=3'b000;

//block13
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;

//block14
    #40 MTN_SENSOR=4'b0100; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;

    #40 ONOFF=2'b10; MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 ONOFF=2'b00; WLL=3'b000;
    end


endmodule
```