

In-House Built Bipedal Walking Robot

K. S. Sim, Y. K. Lum, C. P. Tso

Faculty of Engineering and Technology, Multimedia University

Jalan Ayer Keroh Lama, 75450 Bukit Beruang, Melaka, Malaysia

Email: kssim@mmu.edu.my

Abstract

In this project, an in-house built bipedal walking Robot uses two direct current gear motors to power its legs. Each leg could bend at the knee to assist the walking routines. In order to improve the walking step, a stabilization system is designed through a mechanical and field programmable gate array controller to improve the walking condition. In addition, the robot is able to prevent hitting obstacles while walking.

Keywords: Bipedal Robot, Stabilization, Field Programmable Gate Array, Sensor.

1. Introduction

Biped robot control problems have been paid attention for the past few decades. In the earlier 1980, many researchers derived simple control algorithms and model for biped robots to satisfy the real-time computational requirement [1, 2].

All these approaches have the advantages to simplify the control system design. However, it is still faced problems to derive a proper reduction model. As the problem of real-time control can be solved by the rapid development of computer and VLSI technology, many researchers have applied computer torque method [3], variable structure control [4], fuzzy systems [5], and neural networks [6] to design controllers for biped robots based on the original dynamical equations.

Furthermore, theoretical studies have resulted in numerous publications in many aspects of the subject including kinematics and dynamic modeling, control mechanism, and stability problems [7-8]. Due to the complexity of both theory and practice, however, few biped robots have been built.

In 1993, Kato *et al.* built one of the earliest biped robots, which started to walk [9]. Later on, Kato *et al.* developed a locomotion gait for bipeds referred to as quasi-dynamic walking [10]. Later, the dynamic walking of a biped robot was realized by the same group of scientists [11].

Miura and Shimoyama built a biped that employed a dynamic walking gait [12]. The robot had no ankle torques and each foot contacted the ground at a single point. Raibert then built a two-legged machine [13]. It is similar to Miura's machine and it tips all the time to keep a dynamic balance. Later on, Katoh and Mori [14] built a physical biped robot. The robot was similar to the one built by Miura and Shimoyama. It was then emphasized that a control method of dynamic locomotion was applied to give an asymptotic stability of the trajectory. From the research studies, in some cases, the degrees of freedom of a biped robot became larger than the number of actuators during dynamic walk. To stabilize the biped robot, the singular perturbation technique was applied to control the biped motion.

The purpose of this paper is to propose an Altera Nios Field Programmable Gate Array (FPGA) controller. The Altera Nios board is then incorporated with sensors fixing around the robot. With this mini prototyping robot, a feasible walking robot that can assist patients to avoid obstruction is designed.

2. Physical Design

The mechanical structure of in-house built Biped walking robot (Figure 1) is named as MMU1. The body of the robot measures 15 cm x 12 cm x 20 cm length, width, and height. The foot pad is added with additional aluminum material with dimension 5cm x 5cm to secure the robot while walking. The robot weighs approximately 0.5 kg. Most of the weight is contributed by the gear, twin head motor gearbox, and battery. The motor (as shown in Figure 2) is a dual drive train system that features two powerful DC motors built side-by-side into a gearbox.

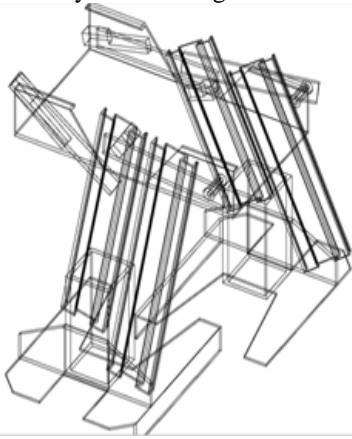


Figure 1. The in-house built biped walking robot



Figure 2. The twin Motor gearbox

3. Circuit Block Diagram

The details of the circuit block diagram are shown in Figure 3. The block diagram consists of H-Bridge motor driver, motor, and piezo speaker for communication.

In this project, the Altera Nios FPGA board is used. This Nios bipedal robot controller circuit is driven by Nios Cyclone chip. The Quartus II Design Software allows to process multi-million gate designs, streamline development flows.

For the H-bridges, they are used to control the two DC motors contained in the gearbox that drives the legs. The left motor drives the leg mechanism on the left side of the robot body, and the right motor drives the legs on the right side of the body. By coordinating the movement of each of the legs using H-Bridges, the robot is capable of walking forward and backward, and turning to the left or right. The structure of the H-Bridges is shown in Figure 4. The details of function can refer to table 1.



Figure 3. The Altera Nios FPGA Board

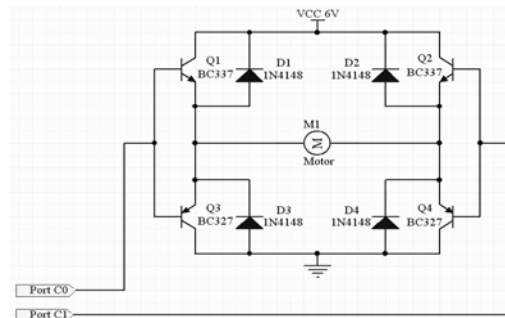


Figure 4. The H-Bridge Circuit

Table 1. The look-up table for the H-Bridge circuit

High Side Left	High Side Right	LowerLeft	LowerRight	Quadrant Description
On	Off	Off	On	Motor goes Clockwise
Off	On	On	Off	Motor goes Counter-clockwise
On	On	Off	Off	Motor "brakes" and decelerates
Off	Off	On	On	Motor "brakes" and decelerates

When turn on the high side left and lower right circuits, the power flows through the motor in the forward direction. For the reverse direction, the high side right and lower left circuits are turned on and power flows through the motor in reverse direction. If turning on the two upper circuits, the motor resists turning and it acts as a breaking mechanism. The same is true if turning on both of the lower circuits. If the terminals of the motor are connected (shorted), then the voltage counteracts the motors from turning.

4. Leg Position Monitoring Circuit

In order to improve the stabilization of robot walking, we program the Nios Board to interface with the additional circuitry namely ADC and potential meter. This process is helped to calibrate the robot (as shown in Figure 5).

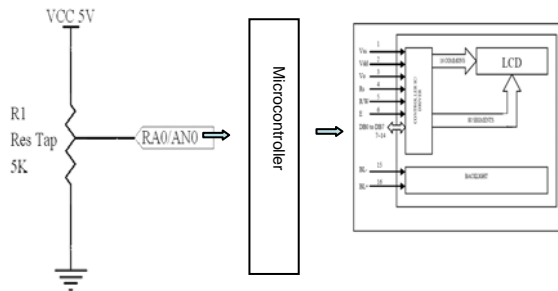


Figure 5. The leg position monitoring circuit



Figure 6. The obstacle detector

5. Obstacle Detector

In this project, we use obstacle sensors. The main purpose of this sensor is to allow the robot to avoid obstruction. SHARP GP2D12 is an infrared distance measuring sensor. It can accurately determine a range between 10 cm and 80 cm. It is an integrated sensor with IR emitter, detector, optics, and timing logic. Its output is an analog voltage and is proportional to the distance to the nearest object in its field of view. It can be used as a proximity detector to detect objects between 0 cm and 130 cm. The output signal is compatible with Nios Board through ADC circuits. The interface is a 3-wire with power, ground and the output voltage. The physical structure of the sensor is shown in Figure 6.

6 Sequence Flow Chart

The sequence of the program begins with self-run test. It operates before the robot starts walking routines. It checks the drive mechanism and sensors to make sure they all function properly. If not, diagnostics and calibration are needed. The sequence then proceeds to the run mode which is walking routine. The detail of the sequence flow chart of the MMU1 robot is shown in Figure 7.

Figure 8 shows the sequence of robot moving forward direction and the detailed of the written program is shown in Appendix.

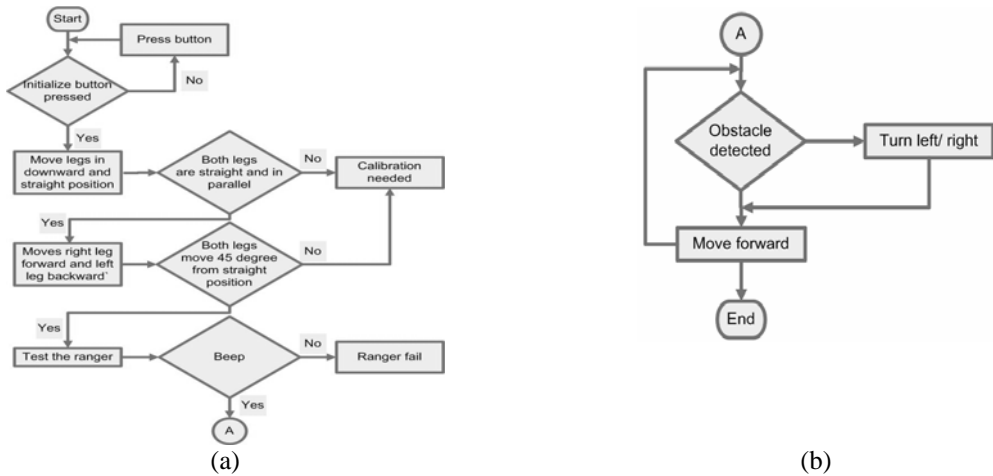


Figure 7. The sequence flow chart of the MMU1 robot (a) and (b).

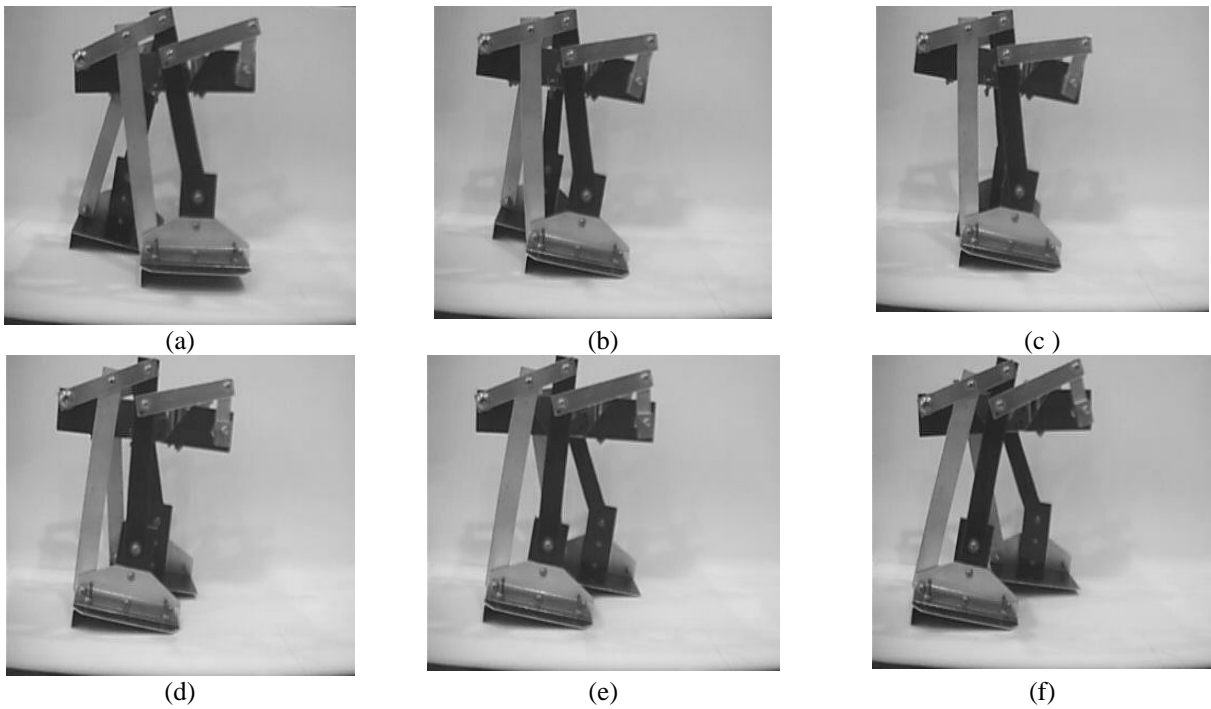


Figure 8. The sequence of robot moving forward

7. Conclusion

In this project, a small-scale, autonomous, bipedal walking robot with an Altera Nios Cyclone chipset has been built. In addition, the robot is able to avoid obstructions through the use of external sensors.

REFERENCES

- [1] Furusho, J., and Masubuchi, M. "A theoretically motivated reduced order for the control of dynamic biped locomotion", Trans. ASME J. Dyn. Sys. Meas. Control, 109, 155--163, 1987.

- [2] Furusho, J., and Sano, A.: ‘Sensor-based control of a nine-link biped’, *Int. J. Robot. Res.*, 9(2), 83-98, 1990.
- [3] Yang, J.S. “Control of a five-link biped using an adaptive inverse dynamics method”, *Control Comput.*, 25(2), 56—63, 1997.
- [4] Tzafestas, S.G., Krikochoritis, T.E., and Tzafestas, C.S. “Robust sliding mode control of nine-link biped robot walking”, *J. Intel. Comput.*, 25(2), 56--63, 1997.
- [5] Juang, J.G. “Fuzzy neural networks approaches for robotic gait synthesis”, *IEEE Trans Syst. Man Cybern. B, Cybern.*, 30(4), 594—601, 2000.
- [6] Miller, W.T. “Real-time neural network control CMAC of a biped walking robot”, *IEEE Control Syst. Mag.*, 14(1), 41—48, 1994.
- [7] Vukobratovic M., Frank A.A, and Juricic D. “On the stability of biped locomotion,” *IEEE Trans. Biomed. Eng.*, vol. BME-17, no. 1.
- [8] Hemami H. “Reduced order model for biped locomotion,” *IEEE Trans. Syst. Man Cyber.*, vol. SMC-8, no. 4, 321--325, 1978.
- [9] Kato I., Ohteru S., Kobayashi H., Shirai K., and Uchiyama A. “Information-power machine with senses and limbs,” in *Proc. 1st CISM-IFTToMM Symp. Theory and Practice of Robots and Manipulation*. New York: Springer-Verlag, 1974.
- [10] Kato T., Takanishi A., Ishikawa H., and Kato I. “The realization of quasi-dynamic walking by a biped walking machine,” in *Proc. 4th CISM-IFTToMM Symp. Theory and Practice of Robots and Manipulation*, PWN, Warsaw, 1983, 341--351.
- [11] Takanishi A., Ishida M., Yamazaki Y., and Kato I. “The realization of dynamic walking by the biped walking robot WL-lORD,” in *Proc. 1985 Int. Conf. Advanced Robotics*, Tokyo, Japan, Sept. 1985, 459—466.
- [12] Wura H., and Shimoyama I. “Dynamical walk of biped locomotion,” *Int. J. Robot. Res.* 3(2), 60--74, 1984.
- [13] Raibert M. H. “*Legged Robots That Balance*”. Cambridge, MA, MIT Press, 1986.
- [14] Katoh K., and Mori M. “Control method of biped locomotion giving asymptotic stability of trajectory,” *Automatica* 20(4), 450—414, 1984.

Appendix

```

--board: apex/nios
--frequency: 33.33MHz crystal
library ieee;
use ieee.std_logic_1164.all;
entity motor_driver is
port(
    pio_8b : in std_logic_vector(7 downto 0);
    clk, ext_rst : in std_logic;
    motor_en : in std_logic;
    ssd : out std_logic_vector(7 downto 0);
    sw5_control, sw8_halffull : in std_logic;
    l297_home_a : in std_logic;
    l297_home_b : in std_logic;
    l297_enable_a, l297_control_a, l297_cwccw_a : out std_logic;
    l297_clk_a, l297_halffull_a, l297_reset_a : out std_logic;
    l297_enable_b, l297_control_b, l297_cwccw_b : out std_logic;
    l297_clk_b, l297_halffull_b, l297_reset_b : out std_logic);
end motor_driver;

architecture nios_motor_driver of motor_driver is
    signal sig_a, sig_b : std_logic;
    signal l297_clk_a_sig, l297_clk_b_sig : std_logic;
    signal speed_1, speed_2, speed_3, speed_4, speed_5, speed_6 : std_logic;
    signal speed_7, speed_8, speed_9, speed_10, speed_11, speed_12 : std_logic;
    signal speed_13, speed_14, speed_15 : std_logic;
begin
    --pio_8b format:
    --pio_8b(7): enable bit for both motor a and b
    --pio_8b(6 downto 4): direction bit,
    --
    : "000">backward, "101">right, "110">left, "111">forward

```

```

--
positive y direction (including left & right) : pio_8b(6) actually is the cw or ccw of the motor, 1 is
--
negative y is downward. : 0 is negative direction (backward). positive y is upward,

```

```

--pio_8b(3 downto 0): speed of the motor

```

```

l297_reset_a <= ext_rst;
l297_reset_b <= ext_rst;
l297_enable_a <= pio_8b(7) and motor_en; l297_enable_b <= pio_8b(7) and motor_en;
l297_control_a <= sw5_control; l297_control_b <= sw5_control;
l297_cwccw_a <= pio_8b(6); l297_cwccw_b <= pio_8b(6);
l297_halffull_a <= sw8_halffull; l297_halffull_b <= sw8_halffull;
--right motor
l297_clk_a <= l297_clk_a_sig and motor_en; --to shutdown the l297 internal translator
l297_clk_a_sig <= sig_a when (pio_8b(6 downto 4) = "000") else --backward
sig_a when (pio_8b(6 downto 4) = "101") else
--right
'0' when (pio_8b(6 downto 4) = "110") else
--left
sig_a when (pio_8b(6 downto 4) = "111") else
--forward
'0';

```

```

--wrong signal

```

```

--left motor
l297_clk_b <= l297_clk_b_sig and motor_en; --to shutdown the l297 internal translator
l297_clk_b_sig <= sig_b when (pio_8b(6 downto 4) = "000") else --backward
'0' when (pio_8b(6 downto 4) = "101") else
--right
sig_b when (pio_8b(6 downto 4) = "110") else
--left
sig_b when (pio_8b(6 downto 4) = "111") else
--forward
'0';

```

```

--wrong signal

```

```

-----ssd-----

```

```

ssd <= "10001110" when ((pio_8b(6 downto 4) = "111") and (pio_8b(7) = '1') and (motor_en = '1')) else
--forward
"10000000" when ((pio_8b(6 downto 4) = "000") and (pio_8b(7) = '1') and (motor_en = '1'))
else --backward
"10001000" when ((pio_8b(6 downto 4) = "101") and (pio_8b(7) = '1') and (motor_en = '1'))
else --right
"11000111" when ((pio_8b(6 downto 4) = "110") and (pio_8b(7) = '1') and (motor_en = '1'))
else --left
"10111111" when ((pio_8b(7) = '0') or (motor_en = '0')) else
"11111111";

```

```

sig_a <= speed_1 when (pio_8b(3 downto 0) = "0001") else
speed_2 when (pio_8b(3 downto 0) = "0010") else
speed_3 when (pio_8b(3 downto 0) = "0011") else
speed_4 when (pio_8b(3 downto 0) = "0100") else
speed_5 when (pio_8b(3 downto 0) = "0101") else
speed_6 when (pio_8b(3 downto 0) = "0110") else
speed_7 when (pio_8b(3 downto 0) = "0111") else
speed_8 when (pio_8b(3 downto 0) = "1000") else
speed_9 when (pio_8b(3 downto 0) = "1001") else
speed_10 when (pio_8b(3 downto 0) = "1010") else
speed_11 when (pio_8b(3 downto 0) = "1011") else
speed_12 when (pio_8b(3 downto 0) = "1100") else
speed_13 when (pio_8b(3 downto 0) = "1101") else
speed_14 when (pio_8b(3 downto 0) = "1110") else
speed_15 when (pio_8b(3 downto 0) = "1111") else
'0';

```

```

sig_b <= speed_1 when (pio_8b(3 downto 0) = "0001") else
speed_2 when (pio_8b(3 downto 0) = "0010") else
speed_3 when (pio_8b(3 downto 0) = "0011") else
speed_4 when (pio_8b(3 downto 0) = "0100") else
speed_5 when (pio_8b(3 downto 0) = "0101") else
speed_6 when (pio_8b(3 downto 0) = "0110") else

```

```

        speed_7 when (pio_8b(3 downto 0) = "0111") else
        speed_8 when (pio_8b(3 downto 0) = "1000") else
        speed_9 when (pio_8b(3 downto 0) = "1001") else
        speed_10 when (pio_8b(3 downto 0) = "1010") else
        speed_11 when (pio_8b(3 downto 0) = "1011") else
        speed_12 when (pio_8b(3 downto 0) = "1100") else
        speed_13 when (pio_8b(3 downto 0) = "1101") else
        speed_14 when (pio_8b(3 downto 0) = "1110") else
        speed_15 when (pio_8b(3 downto 0) = "1111") else
        '0';
--P = f(crystal)/(2*f(desired)) -- f(crystal) = 33.33MHz
--30Hz: p = 555,500
--40Hz: p = 416,625
--50Hz: p = 333,300
--65Hz: p = 256,385
--80Hz: p = 208,312
--100Hz: P = 166,650
--200Hz: P = 83,325
--300Hz p = 55,550
--400Hz p = 41,663

process(pio_8b(3 downto 0), clk, ext_rst) is
    variable count1 : integer range 0 to 555500; --refer the frequency calculation on the above comments
    variable count2 : integer range 0 to 416625;
    variable count3 : integer range 0 to 333300;
    variable count4 : integer range 0 to 256385;
    variable count5 : integer range 0 to 208312;
    variable count6 : integer range 0 to 166650;
    variable count7 : integer range 0 to 83325;
    variable count8 : integer range 0 to 55550;
    variable count9 : integer range 0 to 41663;
    variable count10 : integer range 0 to 33330;
    variable count11 : integer range 0 to 27775;
    variable count12 : integer range 0 to 25639;
    variable count13 : integer range 0 to 23807;
    variable count14 : integer range 0 to 22220;
    variable count15 : integer range 0 to 20831;
    constant const1 : integer := 555500; --refer the frequency calculation on the above comments
    constant const2 : integer := 416625;
    constant const3 : integer := 333300;
    constant const4 : integer := 256385;
    constant const5 : integer := 208312;
    constant const6 : integer := 166650;
    constant const7 : integer := 83325;
    constant const8 : integer := 55550;
    constant const9 : integer := 41663;
    constant const10 : integer := 33330;
    constant const11 : integer := 27775;
    constant const12 : integer := 25639;
    constant const13 : integer := 23807;
    constant const14 : integer := 22220;
    constant const15 : integer := 20831;

begin
    if ext_rst = '0' then
        speed_1 <= '0';
        speed_2 <= '0';
        speed_3 <= '0';
        speed_4 <= '0';
        speed_5 <= '0';
        speed_6 <= '0';
        speed_7 <= '0';
        speed_8 <= '0';
        speed_9 <= '0';
        speed_10 <= '0';
        speed_11 <= '0';
        speed_12 <= '0';
        speed_13 <= '0';
        speed_14 <= '0';
        speed_15 <= '0';

```

```

elsif rising_edge(clk) then
  if pio_8b(3 downto 0) = "0001" then
    count1 := count1 + 1;
    if count1 = const1 then
      count1 := 0;
      speed_1 <= not speed_1;
    end if;
  elsif pio_8b(3 downto 0) = "0010" then
    count2 := count2 + 1;
    if count2 = const2 then
      count2 := 0;
      speed_2 <= not speed_2;
    end if;
  elsif pio_8b(3 downto 0) = "0011" then
    count3 := count3 + 1;
    if count3 = const3 then
      count3 := 0;
      speed_3 <= not speed_3;
    end if;
  elsif pio_8b(3 downto 0) = "0100" then
    count4 := count4 + 1;
    if count4 = const4 then
      count4 := 0;
      speed_4 <= not speed_4;
    end if;
  elsif pio_8b(3 downto 0) = "1000" then
    count8 := count8 + 1;
    if count8 = const8 then
      count8 := 0;
      speed_8 <= not speed_8;
    end if;
  else
    count1 := 0; count2 := 0; count3 := 0;
    count4 := 0; count5 := 0; count6 := 0;
    count7 := 0; count8 := 0; count9 := 0;
    count10 := 0; count11 := 0; count12 := 0;
    count13 := 0; count14 := 0; count15 := 0;
    speed_1 <= '0'; speed_2 <= '0'; speed_3 <= '0';
    speed_4 <= '0'; speed_5 <= '0'; speed_6 <= '0';
    speed_7 <= '0'; speed_8 <= '0'; speed_9 <= '0';
    speed_10 <= '0'; speed_11 <= '0'; speed_12 <= '0';
    speed_13 <= '0'; speed_14 <= '0'; speed_15 <= '0';
  end if;
end process;
end nios_motor_driver;

```