University of Florida

Department of Electrical and Computer Engineering

EEL 5666 Intelligent Machines Design Laboratory (IMDL)

Final report: Balancing Robot

By Wiwat Arunruangsisriloet

Instructor: Dr. Keith L. Doty

Table of Contents

Abstract

Balancing robot is designed to balance itself on its 2 wheels, follow sound, and avoid obstacles by using custom made tilt sensor, sound sensor, IR. and bump sensor. All of the sensors are directly connected to Microprocessor MC68HC11E9. The microprocessor will gather all data, choose the highest priority behavior and send command to control the motors.

Since center of gravity of the robot is in the front of the robot, the robot tends to tilt itself forward when it is no movement. To balance the robot, it has to keep the robot move forward or backward depend on its configurations.

Introduction

Balancing robot is a project for IMDL class, spring 97. Designed to keep itself balance on its wheels by using microcontroller Motorola and software IC developed by MIT. This robot has 4 types of sensors and 4 corresponding behavior that are Bump sensor - Collision avoidance, Sound sensor - Follow sound, Tilt sensor - Balance and IR sensor -Collision avoidance.

Objectives of the robot

Objectives of the balancing robot are

1. Balance itself on 2 wheels
2. Follow the sound
3. Avoid collision by IR.
4. Avoid collision by Bump sensor

Components of the robot

The components of the robot are in the table 1.

Component	Amount
Motorola microcontroller, M68HC11EVBU	1
Servo motor	2
Wheel	2
Novasoft expansion board, ME11	1
Potentiometer	1
Mic.	2
Micro-switch	6
Audio amplifier LM. 386	2
IR emitter	3
IR receiver	2

Table 1.

Behaviors of the robot

This robot has 4 behaviors

1. Balancing itself on 2 wheels. We used Tilt sensor that mounted beneath the robot to sense configuration of the robot. Micocontroller will receive this signal and calculate the error signal. Error signal will be sent to the P control and send the output command to motors.
2. Sound following. Signal of the left and right mic will be sent to audio amplifier LM 386 to amplify the signal. Then pass the signal through diode and capacitor to keep the highest signal and send it to the microcontroller.
3. Obstacle avoidance by IR. The nature of infrared beam is always reflect when there is an obstacle in front of it. Therefore if we can use an IR receiver to sense weather it has obstacle or not.
4. Obstacle avoidance by using bump sensors. For this robot, We used 6 micro-switch to sense weather the robot hit anything or not.

Sensor module

This robot has 4 sensors that are Bump sensor, Tilt sensor, Sound sensor and IR sensor. These sensors are mounted in the place according to the following figure.

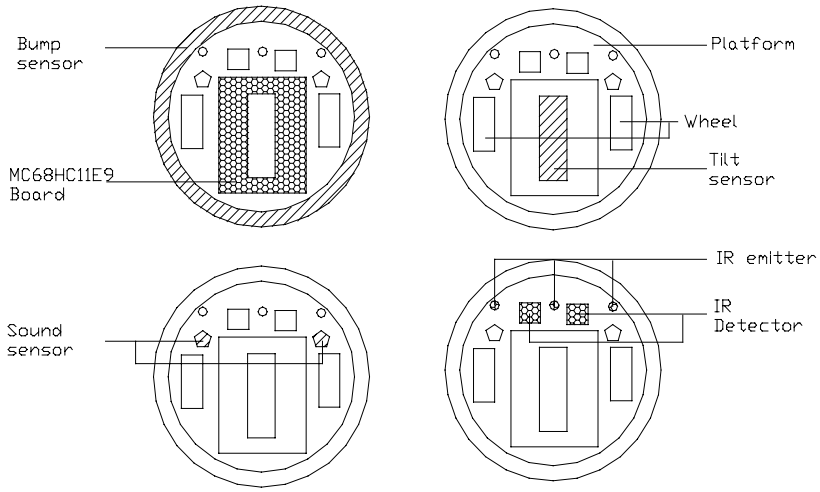

figure 1

IR sensor

IR sensor is compose of 3 IR emitters and 2 IR detectors. IR emitters are connect to port 0x7000 that sends the infrared beam at frequency 40 kHz. When there is an obstacle, The IR beam will be reflected. Then IR detectors can detect the reflected beam.

Tilt sensor

To balance configuration of the robot, it needs efficiency tilt sensor that is reliable, accuracy and cheap. Actually there are a lot of type of tilt sensor in the market. Its price is

among 60 - 400 dollars. Most of them use mercury to sense the configuration. Due to the cost of the sensor is very high, so that we have to build our own tilt sensor.

This sensor use potentiometer to sense configuration of the robot. The components of the sensor are potentiometer and a pendulum. The pendulum is attached to one end of the wire and the other end is attach to the potentiometer. When the robot tilts backward, robot's weight will force pendulum slide backward then the angle of the potentiometer changes. In case the robot tilt forward, the weight of the pendulum will turn potentiometer to the other side. By using the method, we can find the robot configuration by detect the voltage from potentiometer.

This method is very simple, cheap and reliable. It can be implement in every environment and the circuit is very simple.

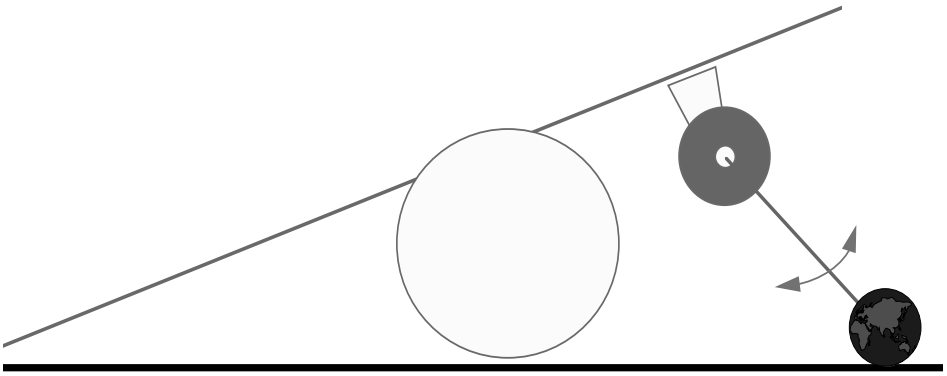

figure 2

From the experiment, we find the relationship between actual angle and the value read from the sensor.

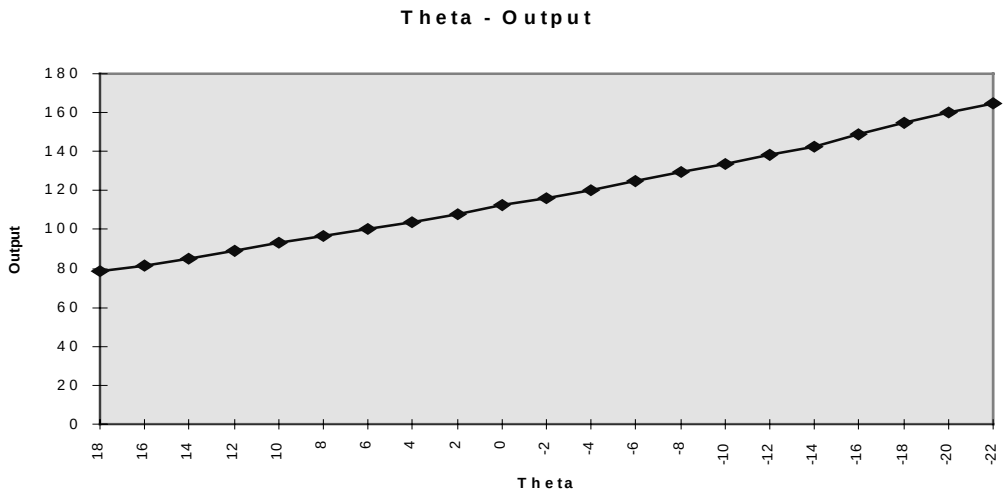

figure 3

Bump sensor

We use 6 micro-switches mounted around the robot as see in the figure

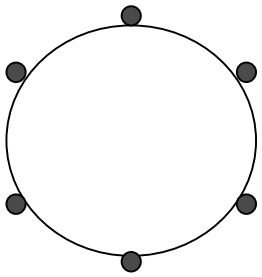

figure 4

Then we connect all the micro-switch as the figure 5

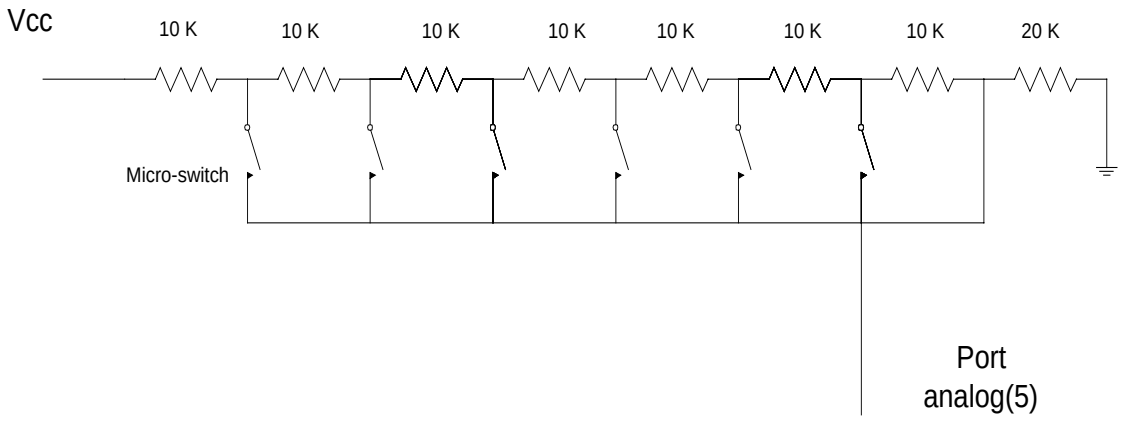

figure 5

Sound sensor

We use 2 Mics directly connect to each audio amplifier, pass through the diode and capacitors to pick keep the highest voltage. Then send this signal to the microcontroller. Mic is mounted on the left side and right side so it can find the direction of sound source.

analog(2) ,
analog(3)

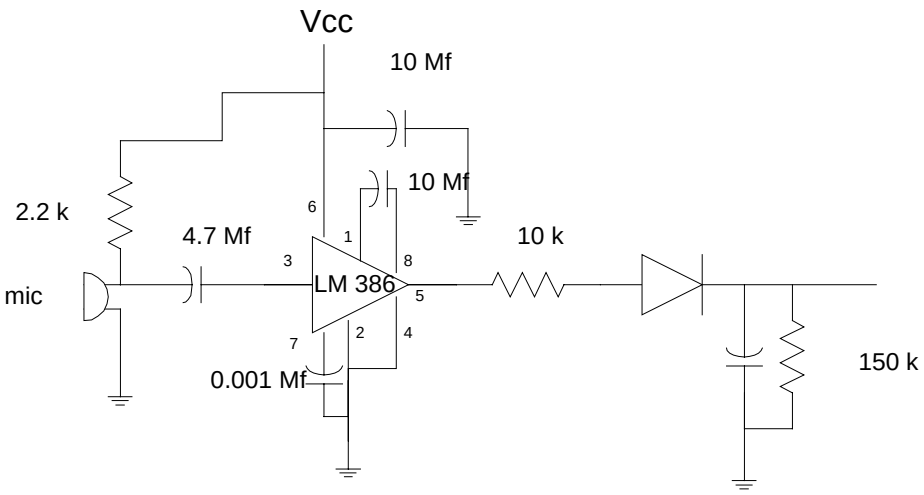

figure 6

Structure of software

Because IC is a multitasking program, we divide each behavior to each function. Functions we have are

1.Sensor function. This function will get all signals from all sensors

2.Bump function. This function will take care of the bump process by bump sensors.

3.Balance function. This function will take care of the balance process.

4.IR function. This function will take care of the bump process by IR.

5.Sound function. This function will take care of the sound following process.

6.Arbitrator function. Arbitrator will pick up all signals for all the sensor function, check the priority of each behavior and then choose the highest one. Since each behaviors have its process, Arbitrator will hold the decision until the process of the previous behavior finish.

7.Motor function. This function will receive command from Arbitrator and send the output to control motors.

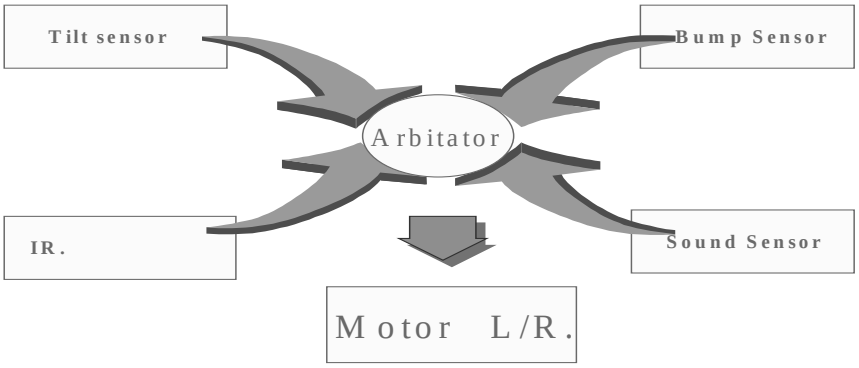

figure 7

Code of this robot is in appendix.

Conclude and Future work

After testing, The robot can balance itself, follow the sound, avoid obstacle by IR and avoid collision by bump sensor. But microcontroller always busy in the balancing process till it can not move forward or backward. From this test, We find some improvement areas which are

1. Sound sensor. Cause the sensitivity of the mic left/right is very different. So we have to check the different voltage every time when we want to run the robot. This problem can be eliminate by building some circuit to compensate the different.
2. For Balancing. Cause we use only P controller. Sometimes, There is a lot of overshoot which can be improve by changing the controller PID controller. And the motors keep working on balancing process so it can not have any forward or backward movement. So we could put another motor to balance the robot instead using the change of velocity to balance the robot.

Appendix

```
/* Program : Balancing robot
   Programmer : Wiwat Arunruangsiriloet
   Date/Time : 04/21/97 17.30
   Purpose : balanace, object avoidance, follow sound
*/

/* declare sensor value */
int ir_r;
int ir_l;
int mic_r;
int mic_l;
int balan;
int bump;
float mot_r_speed;
float mot_l_speed;

/* declare function stage value */
int ir_q;
int mic_q;
int balan_q;
int bump_q;

/*set limit and initial value */
int balan_ini = 103;
int balan_h_lim = 105;
int balan_l_lim = 101;
int ir_r_ini=120;
int ir_l_ini=120;

/* declare speed value */
int b_speed[2];
int m_speed[2];
int i_speed[2];
float bal_speed;

/* declare other parameter*/
int mic_sense;

/* set port */
int ir_emit = 0x07;
int ir_emit_port = 0x7000;
int ir_r_port = 0;
int ir_l_port = 1;
int mic_r_port = 2;
int mic_l_port = 3;
int balan_port = 4;
int bump_port = 5;

/* set other parameter*/
int w1 = 500;
int w2 = 950;
int w3 = 500;
int w4 = 1900;
float gain = 3.50;
int mic_error=15;
int bump_ini = 26;
int bound[] = {30,40,50,90};
int diff = 15;
int slow_move = 15;
long time_start,time_dif;
```

```
void wait(int milisec)
{
   long timer;
   timer = mseconds()+(long)milisec;
   while(timer>mseconds())
      {
      defer();
      }
}


void Fsensor()
{
   while(1)
   {
      ir_r = analog(ir_r_port);
      ir_l = analog(ir_l_port);
      balan = analog(balan_port);
      mic_r = analog(mic_r_port);
      mic_l = analog(mic_l_port);
      bump = analog(bump_port);
   }
}

void Fbalance()
{

  while(1)
     {
     if(balan > balan_h_lim || balan < balan_l_lim )
      {
      balan_q=1;
      bal_speed = ( (float)balan - (float)balan_ini ) * gain;
      if ( bal_speed > 100.0)
        {
        bal_speed = 100.0;
        }
      if (bal_speed < -100.0)
        {
        bal_speed = -100.0;
        }
      time_start = mseconds();
      }
     else
      {
      time_dif = mseconds() - time_start;
      if( time_dif > 1000L )
         {
         balan_q = 0;
         }
      }
     defer();
     }
}
```

```
void Fbump()
{
   while(1)
   {
      if(bump > bump_ini)
       {
       bump_q = 1;
       if(bump > bound[3])
          {
          Fforward(w1,b_speed);
          Fturn_r(w2,b_speed);
          Fforward(w3,b_speed);
          }
       else
          {
          if( bump > bound[2])
             {
             Fforward(w1,b_speed);
             Fturn_l(w2,b_speed);
             Fforward(w3,b_speed);
             }
          else
             {
             if( bump > bound[1])
              {
              Fbackward(w1,b_speed);
              Fturn_l(w2,b_speed);
              Fforward(w3,b_speed);
              }
             else
              {
              if( bump > bound[0])
                 {
                 Fbackward(w1,b_speed);
                 Fturn_r(w2,b_speed);
                 Fforward(w3,b_speed);
                 }
              }
             }
          }
       }
      else
       {
       bump_q = 0;
       }
      defer();
   }
}

void Fforward(int msec,int speed[])
{
   speed[0] = 50;
   speed[1] = 50;
   wait(msec);
}
```

```
void Fbackward(int msec,int speed[])
{
   speed[0] = -50;
   speed[1] = -50;
   wait(msec);
}

void Fturn_r(int msec,int speed[])
{
   speed[0] = 0;
   speed[1] = 50;
   wait(msec);
}

void Fturn_l(int msec,int speed[])
{
   speed[0] = 50;
   speed[1] = 0;
   wait(msec);
}

void Fir()
{
   while(1)
   {
      if(ir_l > ir_l_ini || ir_r > ir_r_ini )
       {
       ir_q = 1;
       if(ir_l > ir_r)
          {
          Fbackward(w1,i_speed);
          Fturn_r(w2,i_speed);
          Fforward(w3,i_speed);
          }
       else
          {
          if(ir_l < ir_r)
             {
             Fbackward(w1,i_speed);
             Fturn_l(w2,i_speed);
             Fforward(w3,i_speed);
             }
          else
             {
             Fbackward(w1,i_speed);
             Fturn_l(w4,i_speed);
             Fforward(w3,i_speed);
             }
          }
       }
      else
       {
       ir_q = 0;
       }
   defer();
   }
}



void Fmic()
{
   float temp;
   while(1)
   {
```

```
      temp = (float)mic_l -(float) mic_r + (float)diff;
      mic_sense = Fabs(temp) ;
      if( mic_sense > mic_error)
       {
       mic_q = 1;
       if ( mic_l > mic_r - diff)
          Fturn_l(0,m_speed);
       else
          Fturn_r(0,m_speed);
       }
      else
       {
       mic_q = 0;
       }
   defer();
   }
}

int Fabs(float val)
{
   int ret_val;
   if( val < 0.0 )
      {
      val = val * -1.0;
      ret_val = (int)val;
      }
   else
      ret_val = (int)val;
   return ret_val;
}
```

```
void Farbitrator()
{
   while(1)
      {
      if(bump_q == 1)
       {
       while(bump_q == 1)
          {
          mot_r_speed=(float)b_speed[0];
          mot_l_speed=(float)b_speed[1];
          defer();
          }
       }
      else
       {
       if(mic_q == 1)
          {
          while(mic_q == 1)
             {
             mot_r_speed=(float)m_speed[0];
             mot_l_speed=(float)m_speed[1];
             defer();
             }
          }
       else
          {
          if(balan_q == 1)
             {
             while(balan_q == 1)
              {
              if(bump_q == 1 || mic_q == 1)
               balan_q = 0;
              mot_r_speed=bal_speed;
              mot_l_speed=bal_speed;
              defer();
              }
             }
          else
             {
             if(ir_q == 1)
              {
              while(ir_q ==1)
                 {
                 mot_r_speed=(float)i_speed[0];
                 mot_l_speed=(float)i_speed[1];
                 defer();
                 }
              }
             else
             {
             mot_r_speed = (float)slow_move;
             mot_l_speed = (float)slow_move;
            }
              }
          }
       }
   }
}

void Fmotor_control()
{
   while(1)
   {
      motor(0,mot_r_speed);
```

```
      motor(1,mot_l_speed);
      defer();
   }
}


void main()
{
   poke(ir_emit_port,ir_emit);
   sleep(.1);
   start_process(Fsensor());
   start_process(Fir());
   start_process(Fbalance());
   start_process(Fmic());
   start_process(Fbump());
   start_process(Farbitrator());
   start_process(Fmotor_control());

}
```