University of Florida

Department of Electrical and Computer Engineering

EEL 5666

Intelligent Machines Design Laboratory

"SOCCER PLAYER ROBOT"

By…

Jun Kunavut

Due 04/28/1997

Instructor: Dr. Keith L. Doty

TABLE OF CONTENTS

1. Abstract

The soccer player robot is a small, mobile robot equipped with a claw and sensors that allow it to detect objects, specify if those objects are obstacle to be avoided or the ball to be hit. If the robot find the ball it will track the ball, try to control it with the claw and take it to the goal, light source. The robot is based on a round wooden platform and is propelled on wheels driven by DC gearhead motors. The robot includes various frequency infrared sensors and photo cells. The behaviors of the robot will allow it to avoid collisions, find the ball, control the ball to the goal. The robot is controlled by a Motorola 68HC11E9 EVBU board with 32k RAM expansion. The controller is programmed with Integrated C.

2. Executive Summary

The soccer player robot is built to imitate the behaviors of a man while playing soccer. The scope of this project is a robot controlled by a Motorola MC68HC11 EVBU board. The robot must be able play soccer by wandering around the room, avoiding the obstacle, finding the ball and controlling the ball to the goal that is light source.

The robot is based on a round wooden platform and is propelled on wheels driven by DC gearhead motors. Two different frequency IR sensors and photo cells are applied to the robot. The four of 40 kHz IR sensors are attached for the obstacle avoidance behavior, and the other one on right claw for detecting if the ball is in the claw. Since we put the 32.8 kHz oscillator in the ball to drive 6 LEDs, another frequency IR sensors, 32.8 kHz IR sensors, are used to detect the Infrared from the ball. For the photo cells, they are used to detect the direction of incoming light. Therefore the robot will be able to specify position of the goal, light source.

Since the MC68HC11 EVBU and its expansion board, ME11 provide the user only 8 analog port but we need 13 analog ports for all sensors mentioned above, an analog expansion was built and attached with the board. Together with the analog expansion, the board now has 15 analog ports.

The current soccer player robot can avoid the obstacles, find the ball in the room, take the ball to the light source and also avoid the obstacle while controlling the ball. Since the surface of the ball is so rough because is inserted through the surface and the ball is not well balanced, the robot cannot well perform the ball controlling. These problems could be solved by changing some sensors and developing the program code.

3. Introduction

At present in the world of considerably advanced technology, it is not necessary anymore that men have to do every task by themselves. We have tried to invent things that would be able to accomplish those kinds of jobs more effectively than that men would. Robot is such a thing as well. A means to develop such a robot is building robots that imitate some behaviors of human that leads into this project, building a soccer player robot. Although we would say that soccer is a game that we like to play by ourselves, this project is not to build a robot to play soccer for us but to practice imitating an activity of man instead. I do believe that this would lead to the invention the working robot in the future.

The scope of this project is to build a soccer player robot. It is obviously seen by the name of the robot that this robot must be able to play soccer. To build a robot that can actually play real soccer, we would spend years and could not get even close to what we have expected. So for this project, I decided to build a robot that can just track the ball, catch the ball, avoid collision and move the ball to the goal (light source).

This paper will introduce to you some details about the MC68HC11 EVBU and ME11, the printed circuit board that is used as the brain of the soccer player robot. Next some figures would help you to imagine how the robot look like. Then we will talk about the motors used for the robot and how to hack them. Sensors is in the next section. Anyway we will not talk in details of each type, but uses of each type of sensors in this project instead. Then behaviors of the robot will be mentioned. From this section, you would understand clearer how the robot plays soccer. Next is the Experiment Layout that was set while building this robot. Finally, the summary will conclude everything accomplished, some problems and future work.

4. Printed Circuit Board Product Line

4.1 ME11: MC68HC11 EVBU Expansion Board

The MC68HC11 EVBU Expansion Board (ME11) allows us to expand the Motorola MC68HC11 EVBU board to realize a complete control oriented microcomputer system. Features include 32 Kbytes of SRAM, motor control, 40 kHz Modulation for IR and sonar, digital inputs, and digital outputs. The ME11 interfaces with the EVBU in a simple and attractive manner. The ME11 does not interfere with the EVBU prototyping area normally reserved for sensors and other interface circuitry (Figure 1).

(Figure 1 from MEKATRONIX DIVISION, NOVASOFT^{TM})

Functional Description of the ME11

The ME11 joined with a Motorola EBBU board realizes a complete microcomputer system with 32 Kbytes of memory. The ME11 stacks underneath the component side of the Motorola EVBU board without interfering with the prototyping area of the EVBU.

The ME11 provides a number of useful functions important to many applications. The ME11 delivers

1) 32 Kbytes of memory (upper 32 Kbytes)
2) an eight-bit digital output unit capable of driving a total of 75 ma. Continuous current where no single output can drive more than 35 ma of continuous current
3) four digital input and four digital output chip enables controlled by the R/W line and the E-clock of the MC68HC11 processor.
4) An H-bridge motor driver that can drive two small DC motors with maximum sustained current of 1 A. through each motor.
5) A stable, crystal-driven, 40 kHz Clock.
6) A 16 pin DIP socket for mounting resistors or other discrete components in series with the digital outputs.
7) A 60 pin header providing direct mating to a connector added to the EVBU board.

ME11's 32 Kbytes of memory provide ample memory for programs and data. Under computer control the eight-bit digital unit can drive and modulate, at 40 kHz, IR emitters, LED's, and other equivalent loads. The 40 kHz signal can also be used to modulate sonar signals. A jumper can disable the 40 kHz and configure the eight digital outputs to serve as computer controlled direct digital outputs. In addition to the eight digital output drivers, ME11 possesses four digital input and four digital output enables that permit the microcomputer to perform a single line enable for four input and four output devices.

figure 2 functional layout of the ME11

Figure 2, from MEKATRONIX DIVISION, NOVASOFT^{TM}, illustrates the principal functional organization of the ME11. The Processor Bus header (J3) provides connections to the ME11 memory, IO, and motor control functions.

The motor speed control derives from Pulse-Width-Modulation (PWM) signal generated by the output compare function on PA5 and PA6. The digital outputs PD4 and PD5 determine the direction of the motors.

The 3:8 decoder provides 4 Input and 4 Output enable signals. Table 1 indicates how the ME11 maps IO into memory. We can refine these addresses by further external decoding using port_B address bits. The latched address bus (U6 output) is not available for address decoding, so the address resolution can get no finer than 256 locations per enable, unless we externally duplicates the generation of the latched addresses.

Table 1 Memory Map of ME11 IO Enable

Name	Direction	Memory Address (Hex)
Y0	Output	$4,000
Y1	Input	$4,000
Y2	Output	$5,000
Y3	Input	$5,000
Y4	Output	$6,000
Y5	Input	$6,000
Y6	Output	$7,000
Y7	Input	$7,000

Figure 3

(From MEKATRONIX DIVISION, NOVASOFT^{TM})

4.2 MSCC11: A Single Chip MC68HC11 Microcontroller

The Mekatronix Single Chip Computer (MCC11), incorporating an MC68HC11 as the on-board processor, is suitable for the project that does not demand extensive computer capability or memory. To communicate code and data between the MSCC11 and a personal computer requires the Mekatronix Bidirectional Serial Communications Board (MBSCB).

Functional Description of the MSCC11

The MSCC11 features an MC68HC11 processor mounted in a 52-pin PLCC and packs tremendous functional capability into a small package. The MSCC11B permits the construction of a single chip microcontroller able to simultaneously control eight 3-wire (5volts, ground, signal) powered analog/digital inputs on Port_E, eight 3-wire powered digital outputs on Port_B and eight 3-wire powered bidirectional digital signals on Port_C. A number of Jumpers provide a variety of options for us. Jumpers can separate unregulated and regulated power and ground rails. We can, for example, employ the unregulated voltage power rail to drive up to 16 servos attached to Port_C and Port_B. The regulated voltage rail always drives the microcontroller and the eight powered digital/analog inputs attached to Port_E. A 5-pin male header permits the MSCC11B to serially communicate with other MSCC11Bs or personal computers via a 5 wire cable to the bidirectional serial communications board (MBSCB).

A minimal configuration of the MSCC11 consists of a microcontroller (MC68HC711E9) in a PLCC socket surrounded by user installed male headers or female connectors with either solder or wire wrap tails to allow wire wrapping external circuits and components to the microprocessor. We need only to install headers required for the application. However, provision has been made to allow all the microcontroller pins to brought out to headers.

4.3 MB2325: Bidirectional Serial Communications Voltage Converter

Serial communication of data and code between a Personal Computer and an embedded microprocessor application requires the conversion of RS-232C voltages to logic voltage levels. This problem is typically solved by placing the voltage conversing circuitry on the microprocessor application circuit board. The embedded application itself, typically, does not require an RS_232C communication port except to download application programs and data or to upload data. Hence, the RS_232C communications circuitry on an embedded microcontroller application unnecessarily occupies valuable board space not relevant to the application itself. Furthermore, any embedded application must also have the same RS-232C support circuitry on the printed circuit board.

The Mekatronix Bidirectional Serial Communications Voltage Converter (MB2325) eliminates this redundant use of RS-232C support circuitry for embedded microprocessor applications by providing the requisite circuitry for RS-232C communications externally. We need only one MB2325 module to perform this task. An MB2325 module permits us to physically connect an RS-232C serial communications cable (restricted to receive (Rx), transmit (Tx), and Data Terminal Ready (DTA) / Data Set Ready (DSR) protocol) to a six wire, 5 volt serial communications cable. Applications include serial communications between a personal computer COM port via an MB23225 to any device, such as the MSCC11, that supports serial communication at logic voltage levels.

Functional Description of the MB2325

The MB2325 permits RS-232c serial communication between a personal computer COM port and any device that provides serial communication at logic voltage levels. The MB2325 converts the personal computer RS-232C voltage levels (-3 to -12 volts active low and +3 to +12 volts active high) to microprocessor logic levels (+5 volts and ground and vice-versa. One blinking LED indicates the presence of a serial data stream from the personal computer to the external, embedded microprocessor application circuit. A second blinking LED indicates the presence of a serial data stream from the external circuitry to the personal computer. When the communications lines are idle, the LEDs shine steadily.

5. Mobile Robot Platform

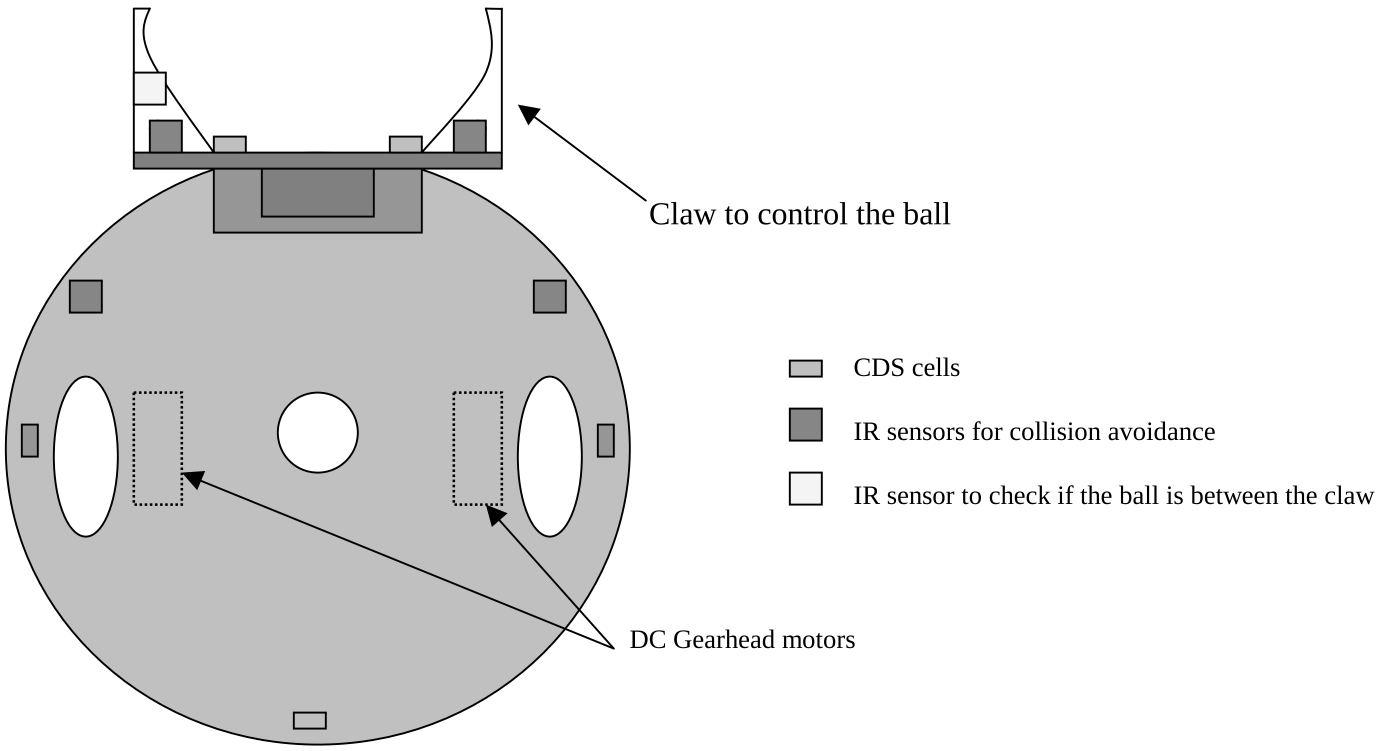

Claw to control the ball
CDS cells
IR sensors for collision avoidance
IR sensor to check if the ball is between the claw
DC Gearhead motors

Figure 4 shows mobile robot platform (Top view)

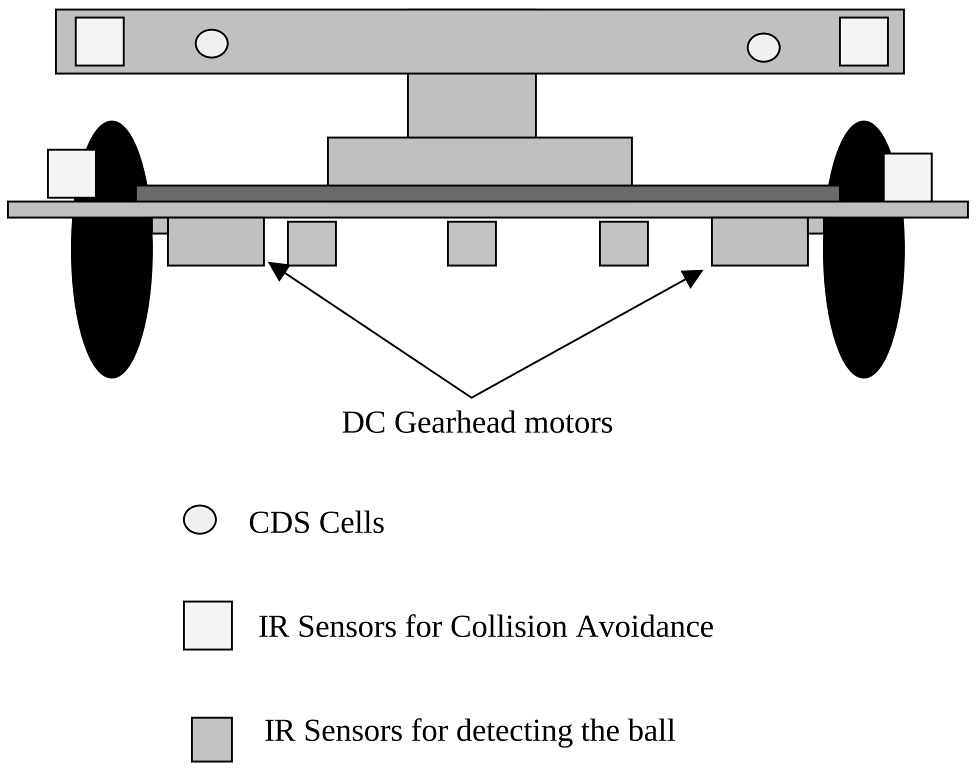

DC Gearhead motors
CDS Cells
IR Sensors for Collision Avoidance
IR Sensors for detecting the ball

Figure 5 shows mobile robot plat form (front view) .

As seen in the figure 4 and 5, the mobile robot platform is round wheeled. We have added the claws, so that the robot would be able to control the ball easier. The platform is made of plywood and has about 2 inches ground clearance.

6. MOTOR AND ACTUATION

We applied only 2 DC gearhead motors for this robot. The objective of these 2 motors is to drive the robot to go forward or backward. We will employ the difference of the speed of these 2 motors to make the robot turn left or right and turn around. The speed of motors is changed up to what behavior the robot is performing.

Those 2 motors are MS410 SERVO hacked as the instruction in the appendices.

7. Sensors

In this project, we will use 3 types of sensors to perform such behaviors we have been talking about. Those sensors are:

7.1 IR sensors.

We use SHARP IR Sensors at two different frequency, 40 kHz and 32 kHz. These sensors are hacked for analog distance measurement as the instruction in the handout from EEL 5666 class, Sharp IR Sensor Hack for Analog Distance Measurement updated 5/4/96 by Keith L. Doty.

7.1.1 SHARP GPIU58Y IR Sensors (40 kHz.)

Four GPIU58Y IR Sensors are attached on the robot for the collision avoidance behavior. As in figure 4 and 5, there are 2 IR sensors attached on the supporter, so it will not detect the ball and confuse the robot if the object in front of the robot is an obstacle or the ball. The obstacles for this project are assumed that they are higher than the ball.

Another GPIU58Y IR Sensor is attached on a claw as in figure 4. This sensor is to check if the ball is caught already.

7.1.2. SHARP GPIU583X IR Sensors (32 kHz)

Three of these sensors on the robot platform are to work together with the LED's and circuitry which are put in the ball. We would mention that this is a sensor suite for the ball-finding and ball-following behaviors. The position of these sensors on the robot platform is shown in figure 5.

7.2 Photoresistors

Because this robot is a soccer player robot, we have to define a goal. We assume for this robot that the goal is a light source. So, a number of photoresistors are used with the robot to specify the position of the goal. We use 5 photoresisters attached on the robot platform as in figure 4 and 5. It is obviously seen that we apply 2 of such these sensors at the front position of the board. This is to increase the precision when the robot moves towards the light source.

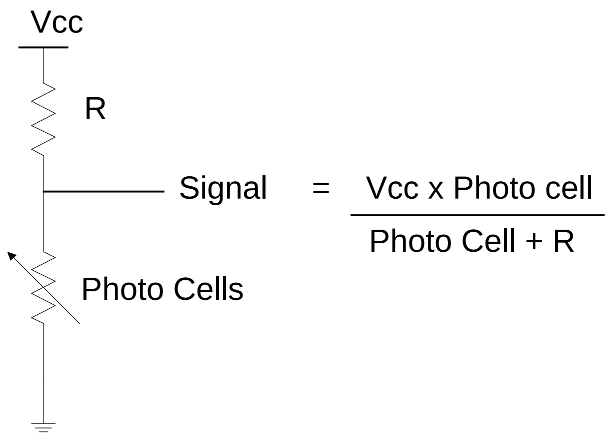


Figure 6 shows the circuitry for Photo cells

From figure 6, we can see that the analog value read from Photo cell is adaptable by changing the resister R to become suitable for the brightness of the room for the robot. For this soccer player robot, we use R = 4.9 kΩ because this value is suitable the brightness of normal room. Anyway, for programming, we didn’t concern about the value read from photo cells because we use the comparison between each photo cell to indicate the direction of light source.

8. The 32.8 kHz Oscillator

For this robot we decided to use 32.8 kHz. IR sensors to detect the ball, so we have to put 32.8 kHz oscillator. This oscillator drives 6 LEDs attached at the surface of the ball. The circuitry of this oscillator can be seen in figure 6

The following is the data of the electronic device on the circuit and the real output of the final circuit that we got from the experiment.

From the figure 6,

RA = 640 Ω

RB = 5.56 kΩ

C = 3.3 ηF.

RL1 = 535 Ω

From equation 4) in the appendices, the frequency of oscillation is :

$$
\begin{aligned}
f &= 1.44 / (RA + 2RB)\,C \\
&= 1.44 / (640 + 2*5.56e3) * 3.3e\text{-}9 \\
&= 37{,}105 \text{ Hz.} \\
&\cong 37 \text{ kHz.}
\end{aligned}
$$

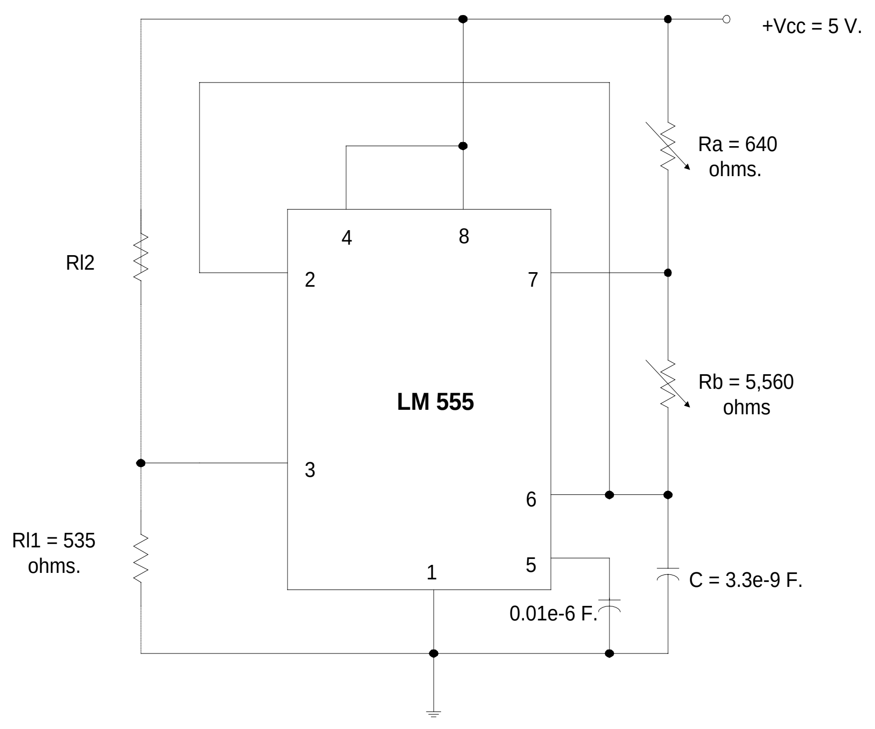


Figure 7 shows the circuitry for oscillator 32.6 kHz.
(The details of the LM555 timer are in the appendices.)

But the value we measured from a multimeter is 32.5 kHz. From equation 5) in appendices, the duty cycle is:

D = RA / (RA+2RB)
= 5.56e3 / (640+5.56e3)
= 0.473

That is close enough to 0.5 that we really want.

9. Analog Expansion

For this robot, we need 13 analog input for all sensors, but the MC68HC11 EVBU Expansion Board (ME11) does provide only 8 analog inputs. Therefore the analog expansion is necessary. The following figure is the circuitry of the analog expansion that is used for this robot. This circuitry gives us 8 analog inputs, but it needs to be connected with one analog input port (PE0). So, we totally have 15 analog inputs.

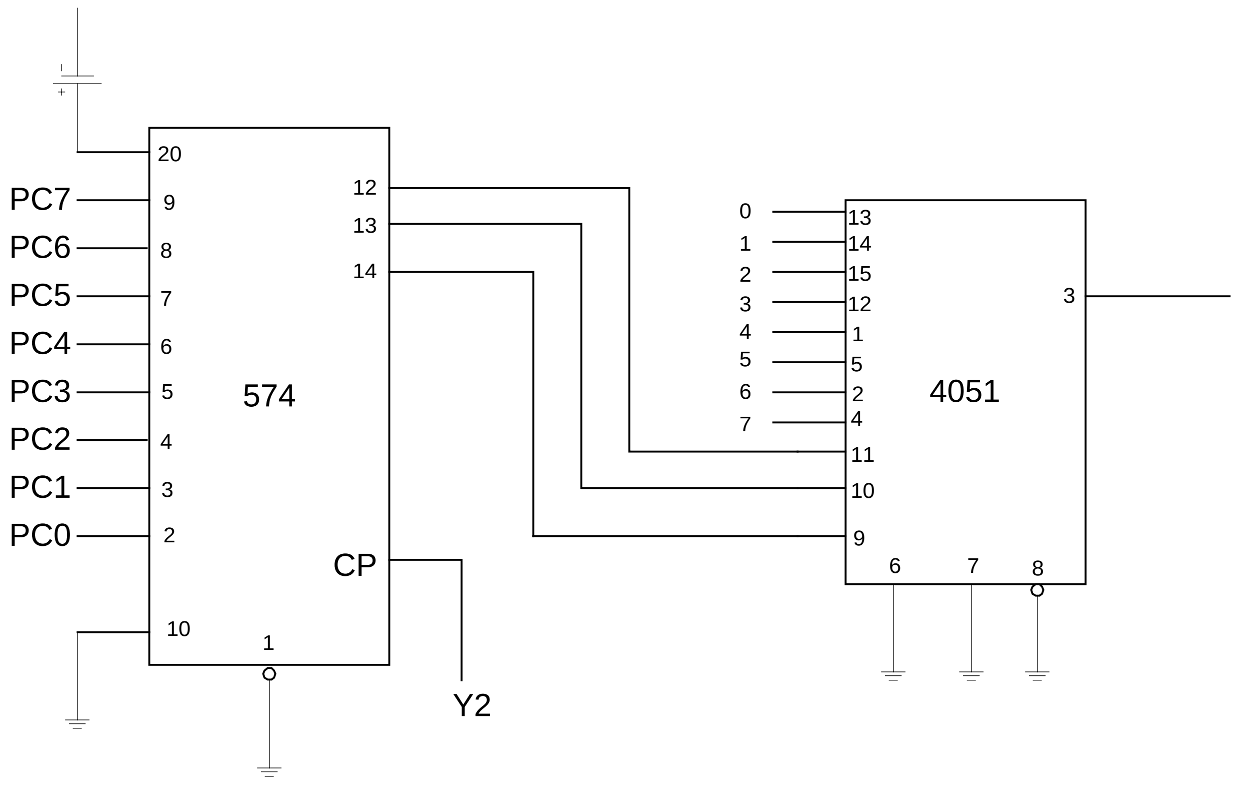


Figure 8 shows the circuitry for analog expansion.

10. Behaviors

1. Ball Tracking Behavior

To find the ball, the robot will move around the room until it sees the ball. Then it will move toward the ball or follow the ball if the ball is moving. This behavior is perform together with the collision avoidance.

2. Collision Avoidance Behavior

The soccer player robot can see the obstacle from 3 direction, left, right and front. If there is an obstacle in the left or right, the robot will turn to another side. To avoid the obstacle in front, we programmed the robot in little more complicated way.

If the robot sees the obstacle in front, it will check from left and right sensors if there is any obstacle on any side. Then it will turn to the side that has no obstacle. If there is no obstacle on both left and sides, the robot will check the distance and position of the object in front to see if it closer to left or right. Then it will turn to another way

.

3. Ball Grasping behavior

Because the ball used with this robot is not well balanced and its surface is not smooth, we have to make a claw for the robot to be able to control the ball forward light source. After the robot has found the ball, it will try to catch the ball before performing light following behavior. The catch succeeds whenever the ball is in the claw and this can be checked by IR sensor attached on the claw.

4. Light following behavior

After the robot grasp the ball, it will check the analog value read from 5 photoresistors to specify the goal that is a light source. Then it will move forward that goal by turning slowly if the goal is not in front of it.

11. The experimental Layout for testing the ability of IR Sensors

For the soccer player robot, tracking the ball is the most important behavior. This experiment is to check how well IR sensor can detect the Infrared light at different positions. In the experiment, I recorded the amount of Infrared (analog value read by IR sensor) from one LED's that can be detected by a SHARP IR Sensor at different position. The data is shown in graph in the figure 8.

In the experiment, without any LED's, the sensor show the value of 85. The maximum value that the sensor could read from the IR emitter was 142. The effective area of the robot's vision is shown in figure ?.

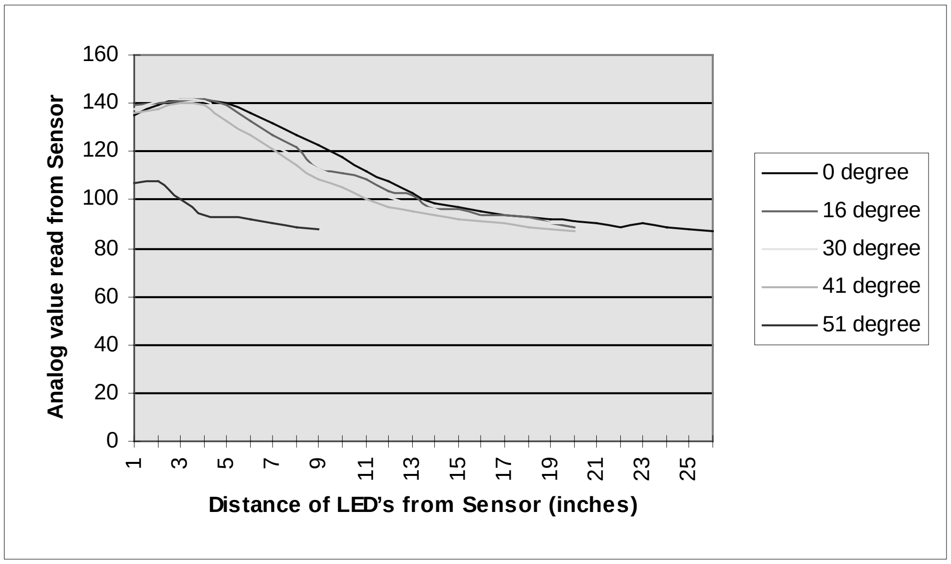
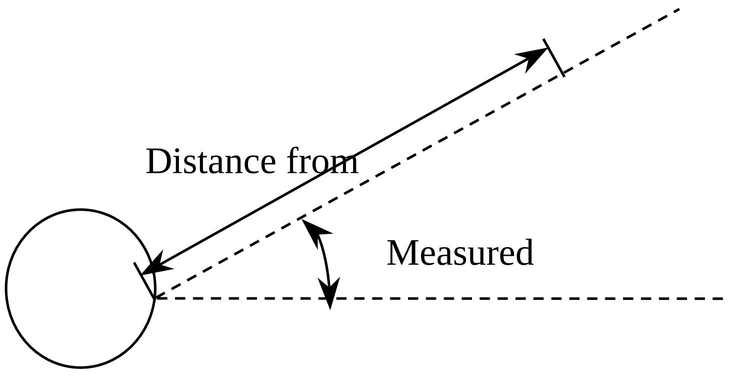


Figure 9 shows the analog value read by sensor at different distance and angle from sensor. The experiment was set up as in the figure 9.



Robot Platform

Figure 10 shows how the experiment was set up.

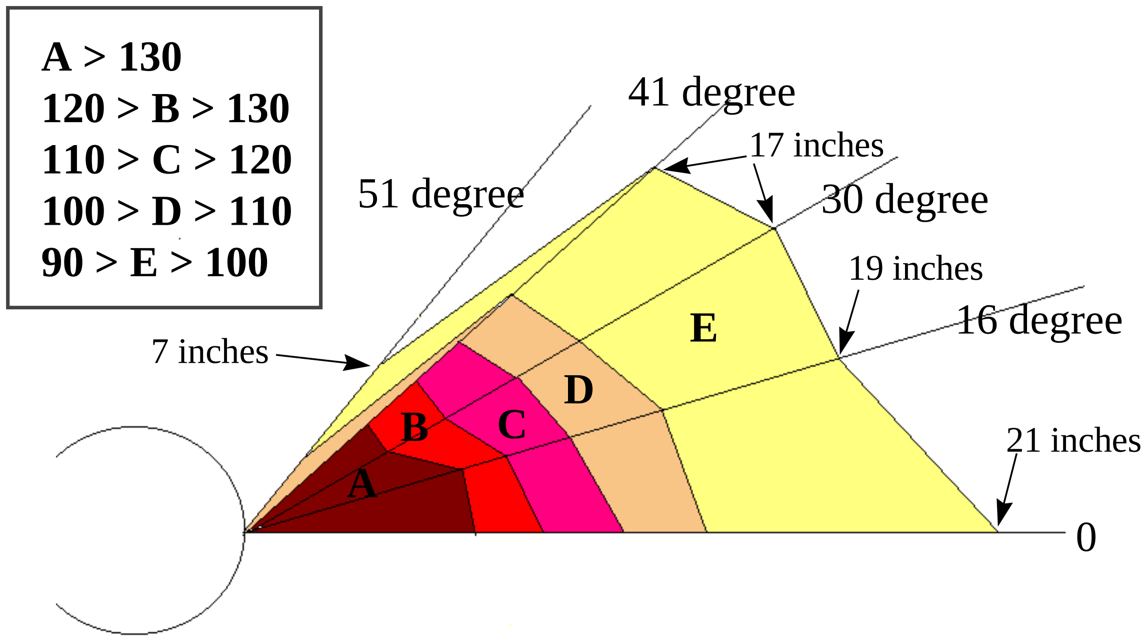


Figure 11 shows the effective range that the robot can detect the Infrared.

12. Conclusion

So far, at the end of Spring Semester 1997, the soccer player robot can perform the tasks as we have mentioned before. I am satisfied with the ball tracking, ball grasping and light following behaviors the robot has performed, but would like to improve the collision avoidance behavior. The robot sometimes failed to avoid the collision with the obstacle or the wall, especially when it tried to catch the ball near the obstacles. Actually I have attached two microswitches at the tip of the claw, so the robot could know when it bumped onto something (this idea is from Professor Doty). I believe that this idea could improve the problem of the collision avoidance behavior, but I couldn't have them to work so far.

During this course, I have experienced many problems from both hardware and software work. Most problems are about lacking experience in Electronics. I often found the mistakes especially about the hardware, and did not know how to debug them. Time was wasted with fixing things that had been done weeks ago. This would be because I did not try to understand things I was doing. If I could start everything over, I would try to understand everything I was going to do. This would shorten time that I had to spend with hardware and then I could have much more time with programming the behaviors or adapting my design later.

For the future work, I would like to add some more sensors, developing some existing behaviors and programming new behavior for the robot to play real soccer. The sensors that I am interested in are IR sensors at different range of frequency. These sensors would be attached on another robot as if it was the opponent. I would change the sensors for detecting the ball to pyroelectric sensors and put some heat source in the ball.

This would help me reduce the problem about the rough surface of the ball and I also could use the 32.8 kHz IR sensors for the opponent of the robot.

For the behavior, I am thinking about adding the ball shooting behavior that let the robot hit the ball so hard when it is close to the goal as if it try to make score in the real soccer game.

13. APPENDICES

A.1 Program Code.

```
/* This program is for “The Soccer player robot”
 • by Jun Kunavut
 • EEL 5666, Intelligent Machine Design Laboratory
 • Spring 1997
 */
float get_ball_left, get_ball_right ;
float ball_find_left, ball_find_right ;
float ball_control_left, ball_control_right ;
float avoid_left, avoid_right ;
float avoid_with_ball_left, avoid_with_ball_right ;
int Ltreshold = 105, Rtreshold = 105 , Ftreshold = 95 ;
int Ltresh = 100, Rtresh = 100, Ftresh = 95 ;
int ball_found = 0 ;
int obstacle = 0 ;
int ready = 0 ;
float right_motor = 50.0, left_motor = 50.0 ; /*for
wandering only*/
int abs(int a)
{
   if(a>=0)
     return a ;
   else
     return -a ;
}

void wait(int milli_seconds)
{
   long timer_a ;
   timer_a = mseconds() - (long) milli_seconds ;
   while(timer_a>mseconds())
     {
      defer() ;
     }
}
void ball_finding()
{
  int left_nose, center_nose, right_nose ;
  float right_motor = 50.0 ;
  float left_motor = 50.0 ;
  while(1)
    {
     left_nose = analog(1) ;
     center_nose = analog(2) ;
     right_nose = analog(3) ;
     if(left_nose < 90 && center_nose < 90 && right_nose <
90)
       {
        ball_found = 0 ;
        ready = 0 ;
        ball_find_left = 50.0 ;
        ball_find_right = 50.0 ;
       }
     else
       {
        ball_found = 1 ;
        defer() ;
       }
    }
}
void avoid()
{
   int Lfront_eye, Rfront_eye, right_eye, left_eye ;
   while(1)
     {
      Lfront_eye = analog(4);
      Rfront_eye = analog(6);
      poke(0x5000, 0x00) ;
      right_eye = analog(0) ;
      poke(0x5000, 0x01) ;
      left_eye = analog(0) ;
      if(Rfront_eye<Ftreshold && Lfront_eye<Ftreshold &&
left_eye<Ltreshold && right_eye<Rtreshold)
        {
         obstacle = 0 ;
        }
      else if((Rfront_eye>Ftreshold) &&
(Lfront_eye<Ftreshold))
        {
         obstacle = 1 ;
         if(left_eye>Ltreshold && right_eye<Rtreshold)
           {
            avoid_right = -50.0 ;
            avoid_left = 50.0 ;
           }
         else
           {
            avoid_left = -50.0 ;
            avoid_right = 50.0 ;
           }
        }
      else if((Lfront_eye>Ftreshold) &&
(Rfront_eye<Ftreshold))
        {
         obstacle = 1 ;
         if(right_eye>Rtreshold && left_eye<Ltreshold)
           {
            avoid_left = -50.0 ;
            avoid_right = 50.0 ;
           }
         else
```

```
           {
            avoid_right = -50.0 ;
            avoid_left = 50.0 ;
           }
        }
      else if((Lfront_eye>Ftreshold) &&
(Rfront_eye>Ftreshold))
        {
         obstacle = 1 ;
         if(right_eye>Rtreshold && left_eye<Ltreshold)
           {
            avoid_left = -50.0 ;
            avoid_right = 50.0 ;
           }
         else
           {
            avoid_right = -50.0 ;
            avoid_left = 50.0 ;
           }
        }
      else if(left_eye>Ltreshold && right_eye<Rtreshold)
        {
         obstacle = 1 ;
         avoid_right = -50.0 ;
         avoid_left =  50.0 ;
        }
      else if(right_eye>Rtreshold && left_eye<Ltreshold)
        {
         obstacle = 1 ;
         avoid_right = 50.0 ;
         avoid_left = -50.0 ;
        }
     }
}
void goaling()
{
   int left_side, right_side, front_side, back_side;
   int Rfront, Lfront ;
   int right = 0 ;
   int left = 1 ;
   poke(0x5000, 0x03) ;
   left_side = analog(0) ;
   poke(0x5000, 0x04) ;
   back_side = analog(0) ;
   poke(0x5000, 0x05) ;
   right_side = analog(0) ;
   poke(0x5000, 0x06) ;
   Rfront = analog(0) ;
   poke(0x5000, 0x07) ;
   Lfront = analog(0) ;
   front_side = (Rfront+Lfront)/2 ;
   while(1)
```

```
     {

if((front_side<left_side)&&(front_side<right_side)&&(front_s
ide<back_side))
           {
            if(abs(Lfront-Rfront)<10)
              {
               ball_control_right = ball_control_left = 50.0
;
              }
            else
              {
               if(Rfront<Lfront)
                 {
                  ball_control_right = 10.0 ;
                  ball_control_left = 50.0 ;
                 }
               else
                 {
                  ball_control_right = 50.0 ;
                  ball_control_left = 10.0 ;
                 }
               }
           }
         else
if((right_side<left_side)&&(right_side<back_side))
           {
            ball_control_right = 10.0 ;
            ball_control_left = 50.0 ;
           }
         else
if((left_side<back_side)&&(left_side<right_side))
           {
            ball_control_right = 50.0 ;
            ball_control_left = 10.0 ;
           }
         else if(back_side<150)
           {
            ball_control_right = 10.0 ;
            ball_control_left = 50.0 ;
           }
      else
        wandering() ;
      poke(0x5000, 0x03) ;
      left_side = analog(0) ;
      poke(0x5000, 0x04) ;
      back_side = analog(0) ;
      poke(0x5000, 0x05) ;
      right_side = analog(0) ;
      poke(0x5000, 0x06) ;
      Rfront = analog(0) ;
      poke(0x5000, 0x07) ;
      Lfront = analog(0) ;
      front_side = (Rfront+Lfront)/2 ;
     }
}
void wandering()
{
   if(left_motor>5.0 && left_motor<50.0)
     {
      right_motor =  30.0 ;
      left_motor =  30.0 ;
     }
   else
     {
      right_motor =  30.0 ;
      left_motor =  30.0 ;
     }
   ball_control_left = left_motor ;
   ball_control_right = right_motor ;
}
int get_ball()
{
   int left_nose, center_nose, right_nose ;
   while(1)
     {
      left_nose = analog(1) ;
      right_nose = analog(3) ;
      center_nose = analog(2) ;
      if(left_nose < 90 && right_nose < 90 && center_nose <
90)
        {
         ball_found = 0 ;
         ready = 0 ;
         defer() ;
        }
      else if((center_nose>right_nose) &&
(center_nose>left_nose))
        {
         get_ball_right = 50.0 ;
         get_ball_left = 50.0 ;
         ball_found = 1 ;
         if(analog(5)<115)
           ready = 1 ;
         }
       else if((left_nose>center_nose) &&
(left_nose>right_nose))
         {
          get_ball_right = 50.0 ;
          get_ball_left = -50.0 ;
          ball_found = 1 ;
          ready = 0 ;
         }
       else if((right_nose>center_nose) &&
(right_nose>left_nose))
         {
          get_ball_left = 50.0 ;
          get_ball_right = -50.0  ;
          ball_found = 1 ;
          ready = 0 ;
         }
    }
}
void ball_controlling()
{
   int left_nose, center_nose, right_nose ;
   while(1)
     {
      left_nose = analog(1) ;
      right_nose = analog(3) ;
      center_nose = analog(2) ;
      if(left_nose < 90 && right_nose < 90 && center_nose <
90)
        {
         ball_found = 0 ;
         ready = 0 ;
         defer() ;
        }
      else if((center_nose>right_nose) &&
(center_nose>left_nose))
        {
         goaling() ;
         ball_found = 1 ;
        }
      else if((left_nose>center_nose) &&
(left_nose>right_nose))
        {
         ready = 0 ;
         ball_found = 1 ;
         defer() ;
        }
      else if((right_nose>center_nose) &&
(right_nose>left_nose))
        {
         ready = 0 ;
         ball_found = 1 ;
         defer() ;
        }
     }
}
void avoid_with_ball()
{
   int Lfront_eye, Rfront_eye, right_eye, left_eye ;
   while(1)
     {
      Lfront_eye = analog(4);
      Rfront_eye = analog(6);
      poke(0x5000, 0x00) ;
      right_eye = analog(0) ;
      poke(0x5000, 0x01) ;
      left_eye = analog(0) ;
      if(Rfront_eye<Ftresh && Lfront_eye<Ftresh &&
left_eye<Ltresh && right_eye<Rtresh)
        {
         obstacle = 0 ;
        }
      else if((Rfront_eye>Ftresh) && (Lfront_eye<Ftresh))
        {
         obstacle = 1 ;
         if(left_eye>Ltresh && right_eye<Rtresh)
           {
            avoid_with_ball_right = 10.0 ;
            avoid_with_ball_left = 60.0 ;
           }
         else
           {
            avoid_with_ball_left = 10.0 ;
            avoid_with_ball_right = 60.0 ;
           }
        }
      else if((Lfront_eye>Ftresh) && (Rfront_eye<Ftresh))
        {
         obstacle = 1 ;
         if(right_eye>Rtresh && left_eye<Ltresh)
           {
            avoid_with_ball_left = 10.0 ;
            avoid_with_ball_right = 60.0 ;
           }
         else
           {
            avoid_with_ball_right = 10.0 ;
            avoid_with_ball_left = 60.0 ;
           }
        }
      else if((Lfront_eye>Ftresh) && (Rfront_eye>Ftresh))
        {
         obstacle = 1 ;
         if(right_eye>Rtresh && left_eye<Ltresh)
           {
            avoid_with_ball_left = 10.0 ;
            avoid_with_ball_right = 60.0 ;
           }
         else
           {
            avoid_with_ball_right = 10.0 ;
            avoid_with_ball_left = 60.0 ;
           }
        }
      else if(left_eye>Ltresh && right_eye<Rtresh)
        {
         obstacle = 1 ;
         avoid_with_ball_right = 10.0 ;
         avoid_with_ball_left =  60.0 ;
        }
      else if(right_eye>Rtresh && left_eye<Ltresh)
        {
         obstacle = 1 ;
         avoid_with_ball_right = 60.0 ;
         avoid_with_ball_left = 10.0 ;
        }
     }
}
void behavior_arbitrate()
{
  int Rwheel = 0 ;
  int Lwheel = 1 ;
  while(1)
    {
     if(!ball_found && !obstacle)
       {
        motor(Rwheel, ball_find_right) ;
        motor(Lwheel, ball_find_left) ;
       }
     else if(!ball_found && obstacle)
       {
        motor(Rwheel, avoid_right) ;
        motor(Lwheel, avoid_left) ;
       }
     else if(ball_found && !ready )
       {
        motor(Rwheel, get_ball_right) ;
        motor(Lwheel, get_ball_left) ;
       }
     else if(ball_found && !ready && obstacle)
       {
        motor(Rwheel, get_ball_right) ;
        motor(Lwheel, get_ball_left) ;
       }
     else if(ready && !obstacle)
       {
        motor(Rwheel, ball_control_right) ;
        motor(Lwheel, ball_control_left) ;
       }
     else if(ready && obstacle)
       {
        motor(Rwheel, avoid_with_ball_right) ;
        motor(Lwheel, avoid_with_ball_left) ;
       }
    }
```

```
}
void main()
{
      start_process(ball_finding()) ;
      start_process(avoid()) ;
      start_process(get_ball()) ;
      start_process(ball_controlling()) ;
      start_process(avoid_with_ball()) ;
      start_process(behavior_arbitrate()) ;
}
```

A.2 LM555/LM555C Timer

General Description

The LM555 is a highly stable device for generating accurate time delays or oscillation. Additional terminals are provided for triggering or resetting if desired. In the time delay mode of operation, the time is precisely controlled by one external resistor and capacitor. For astable operation as an oscillator, the free running frequency and duty cycle are accurately controlled with two external resistors and one capacitor. The circuit may be triggered and reset on falling waveforms, and the output circuit can source or sink up to 200 mA or drive TTL circuits.

Features

- Direct replacement for SE555/NE555
- Timing from microseconds through hours
- Operates in both astable and monostable modes
- Adjustable duty cycle
- Output can source or sink 200 mA
- Output and supply TTL compatible
- Temperature stability better than 0.005% per °C
- Normally on and normally off output

Applications

- Precision timing
- Pulse generation
- Sequential timing
- Time delay generation

- Pulse width modulation
- Pulse position modulation
- Linear ramp generator

Absolute Maximum Ratings

Supply Voltage		+18V
Power Dissipation (Note 1)	LM555H, LM555CH	760 mW
Operating Temperature Ranges	LM555N, LM555CN	1180 mW
LM555C		0°C to +70°C
LM555		-55°C to +125°C
Storage Temperature Range		-65°C to +150°C
Soldering Information		
Dual-In-Line Package		
Soldering (10 Seconds)		260°C
Small Outline Package		
Vapor Phase (60 seconds)		215°C
Infrared (15 Seconds)		220°C

Figure A.1 Schematic Diagram

Figure A.2 Connection Diagram

Application Information

The LM555 Timer actually provides many modes of applications, but we will consider only in the mode of Astable Operation.

Astable Operation

If the circuit as shown in Figure A.3 (pins 2 and 6 connected) it will trigger itself and free run as a multivibrator. The external capacitor charges through RA + RB and discharges through RB. Thus the duty cycle may be precisely set by the ratio of these two resistors.

In this mode of operation, the capacitor charges and discharges between 1/3 Vcc and 2/3 Vcc. As in the triggered mode, the charge and discharge times, and therefore the frequency are independent of the supply voltage. Figure A.3 shows the waveforms generated in this mode of operation.

Figure A.3 shows the application of LM555 in astable mode of operation.

Figure A.4 Shows the waveforms generated in astable mode of operation.

Vcc = 5V

TIME = 20 μS/DIV

RA = 3.9 kΩ

RB = 3 kΩ

C = 0.01 μF

The charge time (output high) is given by:

$$t1 = 0.693(RA + RB) C \quad 1)$$

And the discharge time (output low) by:

$$t2 = 0.693(RB) C \quad 2)$$

-13-

Thus the total period is:

$$T = t1 + t2 = 0.693 (RA + 2RB) C \quad 3)$$

The frequency of oscillation is:

$$f = 1/T = 1.44 / (RA+2RB) C \quad 4)$$

Figure A.5 may be used for quick determination of these RC values.

The duty cycle is: $D = RB / (RA + 2RB)$ 5)

Figure A.5 Free Running Frequency

A.3 MS410 SERVO HACK by Keith L. Doty

Instructions :

1. Remove horn from servo output shaft.
2. Remove 4 screws from bottom plate of servo. Take off the bottom plate and set aside.
3. Carefully remove top plastic cover. Draw a sketch of the gear arrangement so you won't forget it.
4. Remove the gears.
5. Pick up the large output gear. The large output gear has a brass bushing. Find the gear stop tab, a small piece of plastic projection out from the gear shaft. Cut this tab off close to the surface of the gear. Scrap any remaining plastic off the surface to make it smooth. Removing the stop tab will allow the output shaft to turn freely.
6. Pick up the servo case. On the top, notice to small motor mounting screws located opposite the output shaft.
7. Remove the motor mounting screws.
8. The potentiometer shaft projects into the output gear and should be visible with the output gear removed. Push the potentiometer shaft against a flat surface to break the electronic PC board and motor away from the case.
9. Carefully remove the board and motor, which are attached as single unit.
10. Unsolder the motor from the PC board.
11. Clip the three wire cable away from the PC board.
12. Keep the motor and the cable. The PC board will not be used.
13. Put the motor back into the servo case. Screw in the motor mounting screws.
14. Strip away and remove the yellow cable.

15. Pull enough of the red and black wire through the rubber cable grommet to reach the motor terminals. Tin and solder the red and black wire to the motor terminals. (doesn’t matter which to which.)
16. Place the rubber grommet back in place and put the bottom plate on.
17. Turn the servo over, keeping the bottom plate in place. Remount the gear train in proper sequence.
18. Cover the gear train with the top cover.
19. While holding the top cover, the servo case and the bottom plate in place, insert the 4 case screws and tighten opposite corners.
20. The hack is complete.

Testing :

Apply a DC voltage (between 5 and 12 volts) to the red and black wires. The motor should run. The geartrain makes a loud noise when it runs, but if you hear a loud, periodic clicking noise, the plastic stop tab may not have been removed completely. In that case you will need to scrap more of the plastic stop tab off the output gear (step 5).

14. References.

1. "Special Purpose Linear Devices Databook", National Semidonductor, 1989 Edition.
2. Handout from EEL 5666 "*Sharp IR Sensor Hack for Analog Distance Measurement"*, *Updated 5/4/96 by Keith L. Doty, Intelligen Machine Design Laboratory*.
3. Handout from EEL 5666 "MS SERVO HACK" by Keith L. Doty.
4. "Mobile Robots", Joseph L. Jones and Anita M. Flynn, A K Peters, Ltd., 1993.
5. "6.270 LEGO Robot Design Competition Course Notes", 6.270 Organizers, The EECS Department, 1994.
6. "Assembly Instructions for ME11 Expansion Board for the MC68HC11 EVBU", NOVASOFT^{TM}/ Mekatronix Division.
7. "Printed Circuit Board Product Line", NOVASOFT^{TM} Mekatronix Division.
8. "Robot Programming", Keith L. Doty and Reid Harrison, Department of Electrical Engineering, University of Florida.