Kelly Gatson

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Robot Report

Table of Contents

Abstract 3
Executive Summary 4
Introduction 4
Integrated System 4
Mobile Platform 5
Actuation 6
Sensors 7
Behaviors 9
Conclusion 9
Appendix A - Code 11
Appendix B - Part Information 24

Abstract

Doozer is designed to act as a guide dog. It implements a two wheel platform with geared motors. It uses IR light to implement collision avoidance algorithms and bump switches to implement obstacle avoidance algorithms. Additionally, voice recognition technology is used to control Doozer's motion.

Executive Summary

Doozer is designed to act as a guide dog. It is controlled by a 68HC11 microprocessor with an ME11 daughter board. The platform consists of two main parts. There is a head which houses the IR sensors and bump switches and a body which contains the 68HC11, batteries, and the voice recognition sensor. This platform has two wheels. Each wheel is powered is by two motors. Two IR detectors are used to implement collision avoidance and an IR cannon, developed by Aamir Qaiyumi, is used to get enough range on the IR emitters. Bump switches are used to implement obstacle avoidance and to provide input to Doozer during its calibration routine. Additionally, a Capsela Voice Command 3000 module is used to implement voice recognition algorithms. The voltages produced by this sensor signal interrupts to occur in the 68HC11. These interrupts cause Doozer to respond to voice commands.

Introduction

Doozer is designed to be a guide dog robot. Doozer leads its master around obstacles and responds to directional voice commands. Also, Doozer can be calibrated to run on either carpet or tile.

Integrated System

A 68HC11 with an ME11 daughter board is used to control Doozer. Two IR sensors are oriented on the top of Doozer's head to measure distances in front of where the robot is

headed in order to ensure that no obstacles are in the path of the robot. The values of these detectors are read by PortE on the 68HC11. The measurements from these two sensors will be used to determine which way the robot should steer to avoid an obstacle. An IR cannon, developed by Aamir Qaiyumi, is used to emit IR light. The output latch on the ME11 provides the required 40kHz modulated signal to the IR cannon. Two bump switches are located on the front of the robot's head and are used as a back up system. If the IR sensors have failed to detect an obstacle and the robot runs into it, the bump sensor will detect this. These bump switches are connected to PortD2 and PortD3 on the 68HC11. Another switch is located on the front of Doozer's body. This switch is used in calibrating the robot. PortA7 on the 68HC11 detects whether or not this switch is depressed. A piezo buzzer is connected to PortA4 on the 68HC11 this buzzer signals the user when Doozer has finished calibrating. Additionally, a microphone and a reset button are attached, by long leads, to the robot. These two devices are used in the robot's voice recognition system. Output pins on the voice recognition chip are connected to PortA0, 1, and 2 and also to PortE on the 68HC11.

Mobile Platform

Doozer is 20" long, 15" high, and 9" wide and made out of wood. This size allows adequate room for all current hardware and foreseeable future modifications. Doozer is comprised of two distinct pieces. Its head contains the IR detectors and emitters, as well as the bump switches. Doozer's body contains the 68HC11 and ME11 board and the drive train as well as the voice recognition sensor.

Actuation

Doozer rolls on two wheels located towards the front of its body and a slider is positioned in the back of its body to provide balance. Two motors control each wheel. The wheels are 5" in diameter and the motors have a speed of approximately one revolution per second. The motors used are standard with 48 oz-in of torque. To get Doozer to roll fast enough so that it is comfortable to walk next to it, the motors are geared approximately 2:1. This results in Doozer moving about 30" a second at full speed. However, this also cuts the torque of the robot in half. In order to regain some torque, two motors drive each wheel. This brings the torque back to 48 oz-in per wheel. The two motors driving the wheel are connected together and then connected to the motor output pins on the ME11.

The drive train is constructed by using a 3" long piece of welder's steel as an axle. A nylon spacer is glued in the center of the wheels and then the welder's steel is slid through the spacer and secured with epoxy. One end of the axle is inserted into a nylon flange in the side of the robot. Then, a 26 tooth pinion gear is secured on the axle. The other end of the axle is inserted through a brace secured to the inside support of the robot and a nylon flange is epoxyed to the axle on either side of the brace to prevent movement of the axle. Then a 60 tooth spur gear is attached to the motor and the motor is secured to the inside support of the robot making sure that the gears mesh.

Sensors

Doozer uses several different sensors. Infrared (IR) sensors detect distance. Using this distance information, Doozer determines the appropriate actions to take in order to prevent its master from injuring himself by walking into a wall. Additionally, Doozer uses two bump switches to detect if it has run into anything. Doozer also has a sensor which allows it to respond to voice commands. Speaking "back it up," "straight," "turn left," or "turn right" result in the robot's appropriate response.

An IR cannon, developed by Aamir Qaiyumi, is used to emit IR light. The left IR detector is connected to PortE1 and the right IR detector is connected to PortE3. If the value of the left detector exceeds its threshold value, Doozer turns right. If the value of the right detector exceed its threshold value, Doozer turn left.

Doozer's bump switches are connected to +5V through a pull up resistor. If the bump switch is depressed, the signal is pulled down to ground. The left bump switch is connected to PortD2 and the right bump switch is connected to PortD3. The switch that is used for calibration is connected to PortA7.

The voice recognition sensor used in Doozer is part of the Capsela Voice Command 3000 toy. The voice recognition system works by responding to an interrupt caused by the voltages on the voice recognition sensor. The input capture interrupt system is initialized to enable interrupts to occur when a rising edge is detected on any of the interrupt capture pins. To do this, three registers on the 68HC11 must be modified. The PACTL register

must be modified to enable pin 31 on the 68HC11 to function as input capture 4 rather than output compare 5. The TMSK1 register must be modified to enable input captures 1 through 4 to cause interrupts. Finally, the TCTL2 register must be modified to allow the interrupts to occur only on rising edges. Table 4 shows the contents of these three registers, in binary, after these modifications have occurred.

PACTL	0000 0100
TMSK1	0000 1111
TCTL2	0101 0101

Table 1

In addition to the three registers noted in table 4, the register TMSK1 will be utilized in the input capture interrupt service routine. TMSK1 will be examined to determine which input capture caused the interrupt to occur. Then the interrupt service routine will react to the input.

Pin 17 from the voice recognition chip is connected to IC1 on the 68HC11. Pin 18 from the voice recognition chip is connected to IC2 on the 68HC11. Pin 12 from the voice recognition chip is connected to IC3 on the 68HC11. Pins 4 and 13 are used in the circuit shown if Figure 1. This circuit is used to differentiate between the commands "turn left" and "turn right." When "turn right" has been spoken, the value of PortE0 is 138 and when "turn left" has been spoken, the value of PortE0 is 129.

Behaviors

Doozer is able calibrate itself, avoid hitting obstacles, maneuver around obstacles it has accidentally hit, and guide its master around a room by responding to voice commands. Specifically, the robot does the following:

1. When Doozer's program is first run, it checks its front bump switch to determine if it is on carpet or tile. Then it calibrates its IR detectors
2. Doozer will ensure that nothing is in its path or the path of its master. If Doozer does detect an obstacle in the path, it will turn to avoid the object.
3. If Doozer does not "see" an obstacle and runs into it, Doozer will back up and turn to avoid the obstacle before continuing.
4. The robot will respond to "straight," "back it up," "turn left," and "turn right" in the expected manner.

Conclusion

Doozer will act as a very limited guide dog. It will guide its master by responding to directional voice commands. However, if the robot detects that following its current path will injure its master, the robot will change directions. Doozer is not able to detect if the path it is following is wide enough for both it and its master to fit through, nor can it detect if the passageway is high enough. Also, the voice recognition chip does not always respond appropriately to the command that is issued.

In retrospect, much time was wasted designing the drive train and, thus, software development was started with only weeks left in the life of the project. If software has

been developed a little earlier, Doozer may have been able to accomplish more of the tasks that were proposed.

Appendix A - Code

```
#include <me11.h>
#include <motor.h>
#include <hc11.h>
#include <analog.h>
#include <mil.h>

/********************** Vector2 ********************************/
extern void _start(); /* entry point in crt11.s */
extern void voice1(), voice2(), voice3(), voice4(), motor0(), motor1();

#pragma abs_address:0xffd6
/* change the above address if your vector starts elsewhere*/

void (*interrupt_vectors2[])() =
        {
        /* to cast a constant, say 0xb600, use
          (void (*)())0xb600
         */
        (void (*)())0x0,  /* SCI */
        (void (*)())0x0,  /* SPI */
        (void (*)())0x0,  /* PAIE */
        (void (*)())0x0,  /* PAO */
   (void (*)())0x0,  /* TOF */
   voice4,  /* TOC5 */
   (void (*)())0x0,  /* TOC4 */
   motor1,  /* TOC3 */
   motor0,  /*  TOC2 */
   (void (*)())0x0,  /* TOC1 */
        voice3,  /* TIC3 */
        voice2, /* TIC2 */
   voice1, /* TIC1 */
        (void (*)())0x0,  /* RTI */
        (void (*)())0x0,  /* IRQ */
        (void (*)())0x0,  /* XIRQ */
        (void (*)())0x0,  /* SWI */
        (void (*)())0x0,  /* ILLOP */
        (void (*)())0x0,  /* COP */
        (void (*)())0x0,  /* CLM */
                _start   /* RESET */
        };

#pragma end_abs_address
/**************************End of Vector2 ************************/
```

```
/****************************** Globals ***************************/
#define LEFT_EYE 1              /* Left forward IR detector */
#define RIGHT_EYE 3                     /* Right forward IR detector */
#define LEFT_EAR 2              /* Left outward IR detector */
#define RIGHT_EAR 4                     /* Right outward IR detector */
#define UP 5                            /* Upward IR detector */
#define DOWN 6                          /* Downward IR detector */
#define STRAIGHTL -70           /* Value for left motor going straight */
#define STRAIGHTR 70            /* Value for right motor going straight */
#define BACKL 80                /* Value for left motor going backward */
#define BACKR -70               /* Value for right motor goin backward */
#define RCTURNLM -90            /* Value for left motor when turning right on carpet
*/
#define RCTURNRM 20                     /* Value for right motor when turning right
on carpet */
#define LCTURNLM -20            /* Value for left motor when turning left on carpet
*/
#define LCTURNRM 90                     /* Value for right motor when turning left on
carpet */
#define RTTURNLM -80                    /* Value for left motor when turning right on
tile */
#define RTTURNRM -50            /* Value for right motor when turning right on tile
*/
#define LTTURNLM 50                     /* Value for left motor when turning left on
tile */
#define LTTURNRM 70                     /* Value for right motor when turning left on
tile */
#define STILL 0                 /* Value for motors when Doozer is not moving */
#define RIGHT_MOTOR 0
#define LEFT_MOTOR 1
/************************** End of globals ****************************/
/****************************** Functions
*********************************/
#pragma interrupt_handler voice1 voice2 voice3 voice4
void voice1();
void voice2();
void voice3();
void voice4();
void beep(void);
void voice(void);
void avoid(void);
void bump(void);
void straight2(void);
void rightcarpet2(void);
void rightcarpet902(void);
```

```
void leftcarpet2(void);
void leftcarpet902(void);
void stay2(void);
void backup2(void);
void righttile2(void);
void righttile902(void);
void lefttile2(void);
void lefttile902(void);
void backup290(void);
/************************ End of functions ******************************/
/************************ Global Variables ******************************/
int THRESHOLD_LF;                       /* Threshold of left forward IR detector */
int THRESHOLD_RF;                       /* Threshold of right forward IR detector */
int THRESHOLD_LO;                       /* Threshold of left outward IR detector */
int THRESHOLD_RO;                       /* Threshold of right outward IR detector */
int THRESHOLD_U;                /* Threshold of upward IR detector */
int THRESHOLD_D;                /* Threshold of downward IR detector */
int DONE;
int SURFACE;                            /* Stores the surface type */
/********************** End of Global Variables ***************************/

/********************************** Main
*************************************/
int main(void)
{
        init_analog();
        init_motors();
        DONE = 0;
        voice();                /* Initialize interrupts for voice commands */

        while(1)
        {
                avoid();
                bump();
                beep();
        }
}
/********************************** End of Main
*******************************/

void straight2()
{
        motor(RIGHT_MOTOR, STRAIGHTR);
        motor(LEFT_MOTOR, STRAIGHTL);
}
void rightcarpet2()
```

```
{
	motor(RIGHT_MOTOR, RCTURNRM);
	motor(LEFT_MOTOR, RCTURNLM);
}
void rightcarpet902()
{
	long i;
	for(i = 0; i < 1e4; i++)
	{
	motor(RIGHT_MOTOR, RCTURNRM);
	motor(LEFT_MOTOR, RCTURNLM);
	}
}
void leftcarpet2()
{
	motor(RIGHT_MOTOR, LCTURNRM);
	motor(LEFT_MOTOR, LCTURNLM);
}
void leftcarpet902()
{
	long i;
	for(i = 0; i < 1e4; i++)
	{
		motor(RIGHT_MOTOR, LCTURNRM);
		motor(LEFT_MOTOR, LCTURNLM);
	}
}
void stay2()
{
	motor(RIGHT_MOTOR, STILL);
	motor(LEFT_MOTOR, STILL);
}
void backup2()
{
	motor(RIGHT_MOTOR, BACKR);
	motor(LEFT_MOTOR, BACKL);
}
void backup290()
{
	int i;
	for(i = 0; i < 5000; i++)
	{
		motor(RIGHT_MOTOR, BACKR);
		motor(LEFT_MOTOR, BACKL);
	}
}
```

```
void righttile2()
{
        motor(RIGHT_MOTOR, RTTURNRM);
        motor(LEFT_MOTOR, RTTURNLM);
}
void righttile902()
{
        int i;
        for(i = 0; i < 2500; i++)
        {
                motor(RIGHT_MOTOR, RTTURNRM);
                motor(LEFT_MOTOR, RTTURNLM);
        }
}
void lefttile2()
{
        motor(RIGHT_MOTOR, LTTURNRM);
        motor(LEFT_MOTOR, LTTURNLM);
}
void lefttile902()
{
        int i;
        for(i = 0; i < 2500; i++)
        {
                motor(RIGHT_MOTOR, LTTURNRM);
                motor(LEFT_MOTOR, LTTURNLM);
        }
}

void bump()
/************************************ Bump
**********************************/
{
        int a;
        int b;

        CLEAR_BIT(PACTL, 0x80);         /* Initialize Port A7 for input */
        DDRD = 0x33;                    /* Initialize Port D2,3 for input */

        a = PORTD;                      /* Read Port D */
        b = PORTA;                      /* Read Port A */

        if (SURFACE == 0)
        {
                if ((a | 0xfb) == 0xfb)         /* Check left bump switch */
                {
```

```
                backup290();
                rightcarpet902();
                straight2();
        }
        else if ((a | 0xf7) == 0xf7)            /* Check right bump switch */
        {
                backup290();
                leftcarpet902();
                straight2();
        }
        else if((b | 0x7f) == 0x7f)                     /* Check middle bump switch
*/
        {
                backup2();
                rightcarpet902();
                straight2();
        }
    }
    else                                                        /* Doozer is on tile */
    {
        if ((a | 0xfb) == 0xfb)         /* Check left bump switch */
        {
                backup290();
                righttile902();
                straight2();
        }
        else if ((a | 0xf7) == 0xf7)            /* Check right bump switch */
        {
                backup290();
                lefttile902();
                straight2();
        }
        else if((b | 0x7f) == 0x7f)                     /* Check middle bump switch
*/
        {
                backup290();
                righttile902();
                straight2();
        }
    }
}
/********************************** End of Bump
**********************************/
void avoid()
/********************************** Avoid
**********************************/
```

```
{
        long i;
        int a;
        int b;

        int V_LF;                               /* Value of left forward IR detector
*/
        int V_RF;                               /* Value of right forward IR detector
*/
        int V_LO;                               /* Value of left outward IR detector
*/
        int V_RO;                               /* Value of right outward IR detector
*/
        int V_U;                                /* Value of upward IR detector */
        int V_D;                                /* Value of downward IR detector */


        if (DONE == 0)                  /* Calibrate */
        {
                long n;
                int d;
                int min;
                int b [4];
                int c [4];

                CLEAR_BIT(PACTL, 0x80);         /* Initialize Port A7 for input */

                stay2();                        /* turn off motors for this routine */
                OUTPUT_LATCH = 0x00;            /* Turn IR cannon initially off */

                /* Calibrate Doozer for the surface it's on */

                for(n = 0; n < 1e4; n++);               /* delay */
                a = PORTA;                                      /* read Port A */
                if ((a | 0x7f) == 0x7f)                  /* if bottom bump switch is pressed
*/
                        SURFACE = 0;                            /* Doozer is on carpet
*/
                else
                        SURFACE = 1;                            /* if not, Doozer is on
tile */

                /* Beep to signal reading was taken */
                beep();

                for(n = 0; n < 1e3; n++);               /* delay */
```

```
        /* Wait for bottom bump switch to be pressed */
        /* before starting forward IR detector calibration */
        a = PORTA;              /* Read Port A */
        while ((a | 0x7f) != 0x7f)      /* Wait until bottom bump switch is
pressed*/
                a = PORTA;              /* Read Port A *

        /* Calibrate forward IR detectors */
        OUTPUT_LATCH = 0x80;    /* Turn IR cannon on */
        for(i = 0; i < 5e2 ; i++);      /* delay */
        THRESHOLD_LF = analog(LEFT_EYE) + 5;
        THRESHOLD_RF = analog(RIGHT_EYE) + 5;

        OUTPUT_LATCH = 0x00;
        for(i = 0; i < 6000; i++);      /* delay */

        /* Beep to signal end of forward looking IR detector calibration */
        beep();

        /* Wait for bottom bump switch to be pressed */
        /* before starting outward IR detector calibration */
        a = PORTA;              /* Read Port A */
        while ((a | 0x7f) != 0x7f)      /* Wait until bottom bump switch is
pressed*/
                a = PORTA;              /* Read Port A *

        /* Calibrate outward looking IR detectors */
        OUTPUT_LATCH = 0x80;    /* Turn IR cannon on */
        for(i = 0; i < 5e2; i++);       /* delay */
        THRESHOLD_LO = analog(LEFT_EAR);
        THRESHOLD_RO = analog(RIGHT_EAR);
        OUTPUT_LATCH = 0x00;    /* Turn IR cannon off */
        for(i = 0; i < 6000; i++);      /* delay */

        /* Beep to signal end of outward looking IR detector calibration */
        beep();

        /* Wait for bottom bump switch to be pressed */
        /* before starting upward IR detector calibration */
        a = PORTA;              /* Read Port A */
        while ((a | 0x7f) != 0x7f)      /* Wait until bottom bump switch is
pressed*/
                a = PORTA;              /* Read Port A */
```

```
		/* Calibrate upward looking IR detectors */
		OUTPUT_LATCH = 0x80;	/* Turn IR cannon on */
		for(i = 0; i < 600; i++);	/* delay */
		THRESHOLD_U = analog(UP) - 10;
		OUTPUT_LATCH = 0x00;
		for(i = 0; i < 6000; i++);	/* delay */

		/* Beep to signal end of upward IR detector calibration */
		beep();

		/* Wait for bottom bump switch to be pressed */
		/* before calibrating downward IR detector */
		a = PORTA;			/* Read Port A */
		while ((a | 0x7f) != 0x7f)	/* Wait until bottom bump switch is
pressed*/
			a = PORTA;		/* Read Port A *

		/* Calibrate downward looking IR detectors */
		for(n = 0; n < 4; n++)
		{
			OUTPUT_LATCH = 0x80;	/* Turn IR cannon on */
			for(i = 0; i < 600; i++);	/* delay */
			b[n] = analog(DOWN);
			OUTPUT_LATCH = 0x00;
			for(i = 0; i < 6000; i++);	/* delay */
		}
		/* The down threshold is the smallest of the read values */
		/* minus 2.  This allows Doozer to ignore the nominal reading */
		/* but detect a reading if the ground moves away */
		min = b[0];
		for(n = 1; n < 4; n++)
		{
			if (b[n] < min)
				b[n] = min;
		}
		THRESHOLD_D = min - 2;
		/* Beep to signal end of downward IR detector calibration */
		beep();

		/* Wait for bottom bump switch to be pressed */
		/* before returning from this routine */
		DONE = 1;
		a = PORTA;			/* Read Port A */
		while ((a | 0x7f) != 0x7f)	/* Wait until bottom bump switch is
pressed*/
			a = PORTA;		/* Read Port A */
```

```
        }
        else
        {

        OUTPUT_LATCH = 0xff;    /* Turn IR cannon on */

        for(i = 0; i < 5e2; i++);       /* delay */

        V_LF = analog(LEFT_EYE);/* Read left forward IR detector */
        V_RF = analog(RIGHT_EYE);       /* Read right forward IR detector */
        V_LO = analog(LEFT_EAR);        /* Read left outward IR detector */
        V_RO = analog(RIGHT_EAR);       /* Read right outward IR detector */
        V_U = analog(UP);               /* Read upward IR detector */
        V_D = analog(DOWN);                     /* Read downward IR detector */
        OUTPUT_LATCH = 0x00;    /* Turn off IR cannon */
        for(i = 0; i < 3e3; i++);       /* delay */


        if (SURFACE == 0)
        {
                if (V_LF >= THRESHOLD_LF)
                        rightcarpet2();
                else if (V_RF >= THRESHOLD_RF)
                        leftcarpet2();
        }

        else                                                    /* Doozer is on tile */
        {
                if (V_LF >= THRESHOLD_LF)
                        righttile2();
                else if (V_RF >= THRESHOLD_RF)
                        lefttile2();
        }
        }
}
/****************************** End of Avoid
*********************************/
void voice()
/********************************** Voice
*********************************/
{
        /* These registers must have these values to initialize the interrupt system */
        int a;
        a = PACTL | 0x04;
        PACTL = a;              /* Enable input capture 4 */
```

```
        a = TMSK1 | 0x0f;
        TMSK1 = a;              /* Enable input capture 1-3 interrupts */

        TCTL2 = 0x55;                   /* IC1,2,3,4 capture on rising edge */

        INTR_ON();
}
/****************************** End of Voice ******************************/
/* These are interrupt service routines for the voice recognition */
/* sensor on Doozer */
/******************************** Voice1 **********************************/
void voice1()
{
        /* ISR for input capture 1 */

        backup2();

        CLEAR_FLAG(TFLG1, 0x04);                                /* Clear flag */

}
/************************* End of Voice1 ********************************/
/******************************** Voice2 **********************************/
void voice2()
{
        /* ISR for input capture 2 */

        straight2();

        CLEAR_FLAG(TFLG1, 0x02);                        /* Clear flag */
}
/****************************** End of Voice2 ******************************/
/********************************** Voice3 ********************************/
void voice3()
{
        /* ISR for input capture 3 */

        int a;
        int i;
        i = 0;

        while(i == 0)
        {
                a = analog(0);

                if (a <= 135)
                {
```

```
                        i = 1;
                        if (SURFACE == 1)
                                lefttile2();
                        else
                                leftcarpet2();
                }
                else if ((a > 135) & (a < 143))
                {
                        i = 1;
                        if (SURFACE == 1)
                                righttile2();
                        else
                                rightcarpet2();
                }
        }

        CLEAR_FLAG(TFLG1, 0x01);                        /* Clear flag */
}
/*************************** End of Voice3 ******************************/
/******************************** Voice4 ********************************/
void voice4()
{
        /* ISR for input capture 4 */
        int a;

        a = PORTA;
        if ((a & 0x01) == 0x01)
        {
                a = PORTD;
                if ((a & 0x04) == 0x00)                 /* If pin 13 is active low high */
                        stay2();
        }

        CLEAR_FLAG(TFLG1, 0x08);                        /* Clear flag */
}
/*************************** End of Voice4 ******************************/

void beep()
/********************************* beep *********************************/
{
        int a;
        int d;
        int n;

        TCTL1 = TCTL1 & 0xf3;
        a = PORTA;
```

```
        d = (a | 0x10);
        PORTA = d;
        for(n = 0; n < 750 ; n++);
        PORTA = a;
}
/***************************** End of beep ********************************/
```

Appendix B - Part Information

The Capsela Voice Command 3000 was purchased from E-toys.

www.etoys.com