4-23-1996

Software Design Control of an Autonomous Vehicle

Monish Sharma

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation
https://louis.uah.edu/honors-capstones/574

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
Software Design Control of an Autonomous Vehicle

Honors Senior Design Project

By
Monish Sharma
CPE senior

April 23, 1996
ABSTRACT

The Hardware Project is an annual competition between regional universities of the South. The competition is part of the SOUTHECON (south east conference) held by the IEEE (Institute of Electrical and Electronic Engineers) usually during the second week of April. This years hardware project involved design of a fully self-propelled, self-powered, autonomous vehicle of a cubic foot in dimension capable of picking up small steel balls on top of bumpers and dropping them of at home base. An important part of the successful working of the car was a reliable navigation software design that controls the actions and maneuvers of the car. The purpose of my senior honors design project was to successfully design and implement a software navigation control system for the U.A.H car.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Software Navigation Control Design</td>
<td>2</td>
</tr>
<tr>
<td>3. Problems Encountered</td>
<td>4</td>
</tr>
<tr>
<td>4. Conclusion</td>
<td>6</td>
</tr>
</tbody>
</table>
# ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figures</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scoring System</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>Software navigation flow diagram</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>A diagram of the game board layout</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>A diagram of the bumper configuration</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>A diagram of the home base</td>
<td>11</td>
</tr>
</tbody>
</table>
1. **INTRODUCTION**

The I.E.E.E Hardware Design better known as the Region 3 “Car Project” is an annual competition between teams of the universities of the south. This year’s Car Project involved the design of a fully autonomous vehicle that was capable of accomplishing an objective while adhering to the rules and guidelines of the competition. The objective was to try to pick up as many of the five steel balls and drop them off at home base. Points were assigned to successful pick up of the ball from the bumpers and dropping off the ball at home base. In addition, the time taken for the run was also factored into the total points gained.

The car would run on a wooden game board which had five cylindrical PVC pipes called “bumpers” each resting a small steel ball on top. Another cylindrical PVC pipe bigger in diameter than the bumpers was designated as the “home base”. The bumpers had an infrared transmitter fitted on the side facing home base. An IR. signal lasting a maximum of thirty seconds was transmitted from these bumpers in a random sequence. The transmission of the source signal was controlled by the game board operator and the sequence of the bumpers that would source this signal was not disclosed. The car would start from home base. On detection of the source signal it would have to navigate towards the IR. sourcing bumper and pick up the ball on top. It would then have to detect the IR. source signal from home base and navigate towards it. On reaching the home base successfully it would deposit the ball. The run would repeat with a different bumper transmitting the signal until all the bumpers have transmitted the signal.
2. SOFTWARE NAVIGATION CONTROL DESIGN

After carefully reading the rules and guidelines of the competition and understanding the operations that our car would have to perform, it became apparent that the software navigation control design could easily be divided into six modules or procedures. Each module would undertake the accomplishment of a specific objective.

The six modules consisted of:

1. Activate front/back sensors
2. Drive forward/backward
3. Sample function
4. Turn Right/Left/Noturn
5. Determine Proximity
6. Dropball function

1. Activate front/back sensors

The sensors on the car were numbered 0 1 2 3. 0 and 1 being on the right and 2 and 3 being on the left. Another set of four sensors were placed at the rear of the car numbered 4 5 6 7. 4 and 5 being on the same side as 0 and 1. 6 and 7 being on the same side as 2 and 3. The sensors 0 and 1 were connected to 4 and 5 respectively. The sensors 2 and 3 were connected to 6 and 7 respectively. Whether the front or back sensors were active was determined by this software module. The functions `fled()` and `bled()` enabled the front and back sensors for sampling by the sampling function respectively.
2. Drive Forward/Backward

At start the front sensors of the car were activated. Once that was done the car could begin moving forward. The function fdrive() accomplished this. At start, the game board operator would turn the bumper on after receiving the signal from the team leader. This module was placed in front of the sample function so that the car would begin its run at the same time the bumper was to be turned on. The cycle time of the car could then be synchronized with that of the game board as closely as possible.

3. Sample Function

The objective of this module was to determine the sensor number with the highest reading. The function would sample the readings on all the sensors two hundred times and then average them out. The IR. sensors were connected to the A/D port of the MC68HC11E9.

4. Turn Right/Left

This module was necessary in order to steer the car towards the bumper transmitting the IR. signal. The turn() function determined if the car was to turn right, left or not turn at all. It did this by seeing which sensors had the highest reading. If the sensor 0 or 1 was the sensor with the highest reading then it called rturn(). If the sensor 2 or 3 was the sensor with the highest reading then it called lturn(). If none of these sensors had the highest reading then it called noturn().

5. Determine Proximity

The rules stated that other than the ball that designated as the one to picked up if any of the other balls was sensed as moved then that would count as disqualification. The
proximity function helped the car to avoid ramming into the bumper. If the proximity function returned either a FPROX or a BPROX then the drive motors was stopped and the car made no turns.

6. Dropball function

This function operates the stepper motor. It raises the arm first thereby causing the drop of the ball and then lowers it into position to pick up another ball. The dropball function gets called when the home base is reached.

3. PROBLEMS ENCOUNTERED IN DESIGN

While testing the code during the development stages certain problems were encountered. The problems were :-

1) The monitoring of variable status during the run of the code.

SOL) This problem was solved by adding a status function. The function could be inserted anywhere the variables were to be determined. The status function proved useful in monitoring the status of the car. For e.g., turn status, drive status, proximity sensor status, front or back sensor enable status etc.

2) Keeping track of bumper time.

SOL) This problem was easily solved by including an interrupt routine that incremented a global variable that was used to keep track of the time elapsed in seconds. This was critical since the bumper transmitted the signal for a maximum of thirty seconds (if the ball is not picked up within that time). The global variable called cycletime was checked to see if the thirty seconds had elapsed.
3) Real time control of steering motor.

SOL) This was not a major concern since we were able to reach the bumper well within thirty seconds. However, the bumper reach time could be decreased by decreasing the constant right/left turn of the steering motor. This problem could probably be solved in many ways. Some that were thought of were :-

1) Using ranges. Here, weights could be used to increase the range between the two extremes of left and right. With a greater range between left and right the steering motor would make no turns if the value of the difference between left and right sensors fell in that range.

2) We could have mechanically limited the turn angle of the steering device controlled by the dc motor. The device already had limits to the angle that it could turn. By decreasing this angle further the car would not have to make extreme right or left turns.

3) We could have used a stepper motor instead of a dc motor to control the steering of the car. Sending step pulses would have caused an accurate amount of turn. However, finding a stepper motor with a fast enough step time would have been the challenge.

4) We could have used pulse width modulation (PWM). Here, we could have send different widths of pulses to cause different degrees of turn angles. Longer widths could be sent initially and as the car approached the bumper the widths could be reduced.
4. CONCLUSION

The development of a software navigation control system for our car gave me a good knowledge of an embedded system application. The control of hardware through software in real time can lead to interesting problems. The software code was tested and the project was completed successfully. The code was written in C on a Small-C compiler. The Small-C compiler is a bare-bones compiler that was given to us by Motorola.
SCORING SYSTEM:

The scoring method consists of two parts. The first consideration is the number of pinballs retrieved and/or deposited: 50 points will be awarded for each pinball retrieved from a bumper (for a maximum of 250 points from this portion). An additional 50 points will be awarded for each ball transported back to and dropped into the receptacle (again, max. 250 points). This gives a possible 500 points if all five pinballs are successfully picked up and deposited into Homebase. The second component to determine scoring is time. A sliding scale will be used to award bonus points for time. The time bonus starts at 300 points; the official time of each run (in seconds) is subtracted from this baseline figure to calculate the actual time bonus.

For example, if a vehicle removes all five pinballs, successfully deposits two, and finishes in two minutes, thirty seconds (150 seconds), the final score for that run is computed as follows:

\[
\begin{align*}
5 \text{ balls retrieved} \times 50 \text{ points/ball} &= 250 \text{ points} \\
2 \text{ balls deposited} \times 50 \text{ points/ball} &= 100 \text{ points} \\
(300 \text{ seconds} - 150 \text{ seconds}) &= 150 \text{ points}
\end{align*}
\]

TOTAL SCORE

= 500 points

In the event that two teams have the same point total, time will be the determining factor. The team with the lowest run-time will win the tie. In the very unlikely event that two teams have the same point total and the exact same run-time, a tie-breaking run will be held.

Pinballs may be removed from atop the bumpers only when the transmitter is on. If at any time a ball is removed or deposited out of the designated sequence, a contest official will stop the run and the competing team will be awarded points earned up until that point of the competition, not including the time bonus.

The time of each run and the number of balls retrieved/returned will be monitored by an automated system and verified by contest officials.

ENFORCEMENT OF COMPETITION RULES:

An official will be present at each PINBALL TABLE to verify the time of each run. A referee will also be present to verify the final score and enforce any and all rules contained in this document. Violation of any of these rules, or behavior of any team member viewed by officials as unsportsman-like will result in disqualification of that team. All rulings by the contest officials are final.
main_loop()
for(cycle=0;cycle<5;cycle++)

Sample()

Fdrive()

Fled()

Initall()

Bdrive()

Bled()

Noturn()

Nodrive()

Dropball()
Figure 1b**: BASIC TABLE CONFIGURATION (showing random bumper alignment)

* Inner dimensions are 3/4" shorter from each end than the outer dimensions, due to the thickness of the boards that comprise the sides of the table.

** All specifications given are to within tolerances of ± 1/4" on the wood (the table) and ± 1/8" on the PVC (Homebase and the bumpers).

*PINBULL WIZARD -- Revision #2 (November 1995)*
Figure 2*: BUMPER CONFIGURATION (top and side views)

* All specifications given to within a tolerance of ± 1/8".
Figure 3*: RECEPTACLE (HOME BASE)

- All specifications given are to within a tolerance of ± 1/8".

PINWHEEL WIZARD -- Revision #2 (November 1995)
This program controls the operations of the MC68HC11E micro-controller. It allows the 68HC11 to control/access the hardware on the car, e.g., IR sensors, steering motor, drive motor, robotic arm, proximity switches, etc. Besides that the program also calculates the time elapsed for each bumper in cycle time.

/* Define where things should reside... */
#define REG_BASE 0x1000

#include <startup.c>
#include <68hc11.h> /* Defines where register banks are mapped... */
#include <stdio.h>

/* Link in the following libraries (nearly all needed by fprintf())...*/
#include <fopen.c>
#include <fputs.c>
#include <fputc.c>
#include <itoa.c>
#include <atoi.c>
#include <is.c>
#include <fprintf.c>
#include <itoab.c>
#include <reverse.c>
#include <strlen.c>
#include <delay.c>

/* Link in the device driver for the Motorola 68HC11 EVB...*/
#include <EVB_out.c>

#define NUM 200 /* Number of samples... */
define DELAY 400
define MAXSTEP 336
define NSTEPS 8
define CLEAR 0x00
define BGND 30 /* approx. vals given by sensors in just Background light */
define WGHT0 -15 /* outer weight are for determining direction to turn left */
#define WGHT1 -7 /* inner weight are for determining direction to turn left */
#define WGHT2 7 /* inner weight are for determining direction to turn right */
#define WGHT3 15 /* outer weight are for determining direction to turn right */
#define MINST -70 /* turn left if left leds < Minst */
#define MAXST 70 /* turn right if the right leds > Maxst */
#define ZONE 20

/* PIA addresses........... */
#define DR_PA 0xc000 /* data reg for port a */
#define CR_PA 0xc001 /* control reg for port a */
#define DR_PB 0xc002 /* data reg for port b */
#define CR_PB 0xc003 /* control reg for port b */

#define BDRIVE 0x18 /* PortB 1 on B4(motor backward) and B3(Drive enable) pins */
#define FDRIVE 0x48 /* PortB 1 on B6(motor forward) and B3(Drive enable) pins */
#define RSTEER 0x22 /* PortB 1 on B5(turn right) and B1(Turn enable) pins */
#define LSTEER 0x82 /* PortB 1 on B7(turn left) and B1(Turn enable) */
#define FLED 0x01 /* PortB 0 on B2(front LED) pin */
#define BLED 0x04 /* PortB 0 on B1(back LED) pin */

#define BIT0 0x01 /* only turns bit 0 on */
#define BIT1 0x02 /* only turns bit 1 on */
#define BIT2 0x04 /* only turns bit 2 on */
#define BIT3 0x08 /* only turns bit 3 on */
#define BIT4 0x10 /* only turns bit 4 on */
#define BIT5 0x20 /* only turns bit 5 on */
#define BIT6 0x40 /* only turns bit 6 on */
#define BIT7 0x80 /* only turns bit 7 on */

#define MAXTIME 915 /* 915 increments = 30sec for a 32.77ms prescalar of 1 */
#define DELAYTIME 50
#define SENABLE 0x80 /* For Stepper Motor */
#define S1 0x02
#define S3 0x08
#define S2 0x10
#define S4 0x40
unsigned int cycletime;
unsigned int bval;   /* keep track of PortB value */
int led[4];
unsigned char ledmax, ledmaxv;
unsigned int bgnd;

unsigned int step[NSTEPS] = {
    SENABLE | S4 | S2,
    SENABLE | S4,
    SENABLE | S4 | S1,
    SENABLE | S1,
    SENABLE | S3 | S1,
    SENABLE | S3,
    SENABLE | S3 | S2,
    SENABLE | S2
};

FILE stdout;

/**********************MAIN()*******************************/
main()
{
    int cycle;
    unsigned int stepnum, i;

    stdout = fopen(EVB_out);

    initall();    /* initialization function */

    cycletime = 0;
    while( cycletime < 150 )
    {
        /* lower the arm */
        for( stepnum = MAXSTEP; stepnum > 0; stepnum--)
        {
            pokeb(DR_PA, step[stepnum % NSTEPS]);
            for( i=0; i<DELAY; i++ )
        }

        for( cycle=0; cycle < 5; cycle++ )
        {
            fled();
        }
    }
}
fdrive();
cycletime = 0;
/* status(); */
while( cycletime < MAXTIME)
{
    sample(); /* sample the ir sensor data digitally */
turn();
    if( proximity() == FPROX)
        cycletime = MAXTIME - DELAYTIME;
}
/* fprintf(stdout, "Bumper reached\n"); */

nodrive(); /* stop the drive motors by breaking */
noturn();
/* ............NOW GOING BACK TO THE HOME BASE......................... */
bled();
bdrive();
cycletime = 0; /* cycle time is initialized */
while( cycletime < MAXTIME)
{
    sample(); /* sample the ir sensor data digitally */
turn();
    if( proximity() == BPROX)
        cycletime = MAXTIME - DELAYTIME;
}
nodrive();
noturn();
/* fprintf(stdout,"Reached Home Base.\n"); */
dropball();
173 }
174 /* ***********************************************************/
175 /* This function initializes all the ports, changes the timer interrupt 
176 vector routine at 0xD1 to point to function resetTOF(), clears the timer 
177 overflow flag & enables the timer overflow interrupt */
178 /*************************************************************************/
179 /* ***********************************************************/
180 initall()
181 {
182     initPA(0xfa); /* initialize porta as inputs for sensor readings */
183     initPB(0xff); /* initialize portb pins as outputs for motor control */
184     bval = CLEAR ;
185     pokeb(DR_PA, CLEAR); /* write 0's to all the output as well as input bits */
186     pokeb(DR_PB, bval); /* write 0's to all the output bits */
187     /* change timer overflow interrupt vector routine at 0xD1 to resetTOF() */
188     poke(0xD1, resetTOF);
189     /* clear timer overflow flag */
190     bit_set(REG_BASE+TFLG2, BIT7);
191     /* enable timer overflow interrupt */
192     bit_set(REG_BASE+TMSK2, BIT7);
193     pokeb( REG_BASE+ADCTL, 0x20 | 0x10 | 0x04 ) ;
194     e_int();
195 }
196 /* ***********************************************************/
197 /* This function is the interrupt service routine which gets called every 
198 time the counter hits 32.77ms for a prescalar of 1. An overflow flag 
199 gets set in TFLG2 reg which needs to be cleared for the counter to 
200 start counting again. */
201 /*************************************************************************/
202 interrupt resetTOF() 
203 {
204     /* increment cycle time (32.77 ms) */
205     cyclertime++ ;
/* reset timer overflow flag */
bit_set(REG_BASE+TFLG2,BIT7);

} /*PROXIMITY()*/

/* This function aids in telling the MC68hc11E9 if the bumper or home base is at proximity */

/****************************PROXIMITY()**************************************/

// This function aids in telling the MC68hc11E9 if the bumper or home base is at proximity */

proximity()
{
    unsigned int proxa;
    proxa = peekb(DR_PA);

    if( proxa & BIT2 ) /* if front proximity switch goes HIGH */
        return FPROX;
    else if( proxa & BIT0 ) /* if back proximity switch goes HIGH */
        return BPROX;
    else
        return NOPROX;

} /*PROXIMITY()*/

/****************************FLED()*****************************************/

/* This function enables the front sensors */

/****************************FLED()*****************************************/

fled()
{
    bval = (bval & ~BLED) | FLED;
    pokeb(DR_PB, bval);

} /*FLED()*/

/****************************BLED()*****************************************/

/* This function enables the back sensors */

/****************************BLED()*****************************************/

bled()
{
    bval = (bval & ~FLED) | BLED;
    pokeb(DR_PB, bval);

} /*BLED()*/

/****************************FDRIVE()*****************************************/

/* This function enables the drive motor to drive forward */

/****************************FDRIVE()*****************************************/
fdrive()
{
    bval = (bval & -BDRIVE) | FDRIVE;
pokeb(DR_PB, bval);
}

/* This function enables the drive motor to drive backward */
/******************************BDRIVE()******************************/

nодrive()
{
    bval = bval & -(FDRIVE | BDRIVE);
pokeb(DR_PB, bval);
}

/* This function makes the drive motor stop */
/******************************NODRIVE()******************************/

rturn()
{
    bval = (bval & -LSTEER) | RSTEER;
pokeb(DR_PB, bval);
}

/* This function makes the steering motor turn right */
/******************************RTURN()******************************/

lturn()
{
    bval = (bval & -RSTEER) | LSTEER;
pokeb(DR_PB, bval);
}

/* This function makes the steering motor turn left */
/******************************LTURN()******************************/

noturn()
/* This function keeps the steering motor from turning */
/******************************NOTURN()******************************/
302 noturn()
303 {
304     bval = bval & -(RSTEER | LSTEER);
305     pokeb(DR_PB, bval);
306 }
307 /* ******************************************TURN()*******************************************/
308 /* This function makes the steering motor turn in the respective direction*/
309 /******************************************************************************************/
310 turn()
311 {
312     /* right turn if the right sensors have greater values than left sensors */
313     if ( (ledmax == 0) || (ledmax == 1) )
314         rturn();
315     /* left turn if the left sensors have greater values than right sensors */
316     else if( (ledmax == 2) || (ledmax == 3) )
317         lturn();
318     else
319         noturn(); /* keep going straight */
320 }
321 /* ******************************************SAMPLE()*******************************************/
322 sample()
323 {
324     unsigned int i;
326     for (i = 0; i < NUM; i++)
327     {
328         /* take four sampled guys out of the ADDR1-4 */
329         led[0] = peekb(REG_BASE + ADR1);
330         led[1] = peekb(REG_BASE + ADR2);
331         led[2] = peekb(REG_BASE + ADR3);
332         led[3] = peekb(REG_BASE + ADR4);
333     }
334     led[0] = led[0]/NUM ;
335     led[1] = led[1]/NUM ;
337     led[3] = led[3]/NUM ;
ledmax = -1;
ledmax = 0;

    ledmax = 0;
    ledmax = 3;
else
    ledmax = -1;

/****************************************DROPBALL()*********************************************/

/* This function raises and then drops the arm after the car reaches the home base. By raising the arm the car manages to drop the ball. Then it positions the arm back in place to pick up another ball from a bumper. */

 /***************************************************************************/
dropball()
{
    unsigned int stepnum;
    unsigned int i;

    for( stepnum = 0; stepnum < MAXSTEP; stepnum++)
    {
        pokeb(DR_PA, step[stepnum % NSTEPS]);
        for( i=0; i<DELAY; i++);
    }

    for( stepnum = MAXSTEP; stepnum > 0; stepnum--)
    {
        pokeb(DR_PA, step[stepnum % NSTEPS]);
        for( i=0; i<DELAY; i++);
    }

} xcb b

/******************************initPA(ioBits)**********************************************/

initPA(ioBits)
{
    unsigned char ioBits;

    /* select portA data direction reg on PIA inorder to make pins ins or outs */
    pokeb(CR_PA, 0x00);
/* sets all bits with value 0 as input pins and those with 1 as output pins */
pokeb(DR_PA, ioBits);
/* select output reg, making it output now */
pokeb(CR_PA, 0x04);
}

/*************************************************************************/
initPB(ioBits)
unsigned char ioBits;
{
/* select portB data direction reg on PIA inorder to make pins ins or outs */
pokeb(CR_PB, 0x00);
/* sets all bits with value 0 as input pins and those with 1 as output pins */
pokeb(DR_PB, ioBits);
/* selects output reg, making it output now */
pokeb(CR_PB, 0x04);
}
/*************************************************************************/
status()
{
/* direction */
if (bval & FDRIVE)
    fputs('Forward', stdout);
if (bval & BDRIVE)
    fputs('Backward', stdout);
if (!(bval & FDRIVE) && !(bval & BDRIVE))
    fputs('Nodrive', stdout);
/* turning */
if ((ledmax == 0) || (ledmax == 1))
    fputs('Rturn', stdout);
if ((ledmax == 2) || (ledmax == 3))
    fputs('Lturn', stdout);
if (ledmax == -1)
    fputs('Noturn', stdout);
/* proximity */
    if (proximity() == FPROX)
        fputs('Front', stdout);
    if (proximity() == BPROX)
        fputs('Back', stdout);
    if (proximity() == NOPROX)
        fputs('None', stdout);
    fputs('\n', stdout);