Date: 4/25/01
Student: Nicholas Ivano
TA: Scott Nortman
Rand Chandler
Instructor: A. A. Arroyo

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

Magellan Final Report

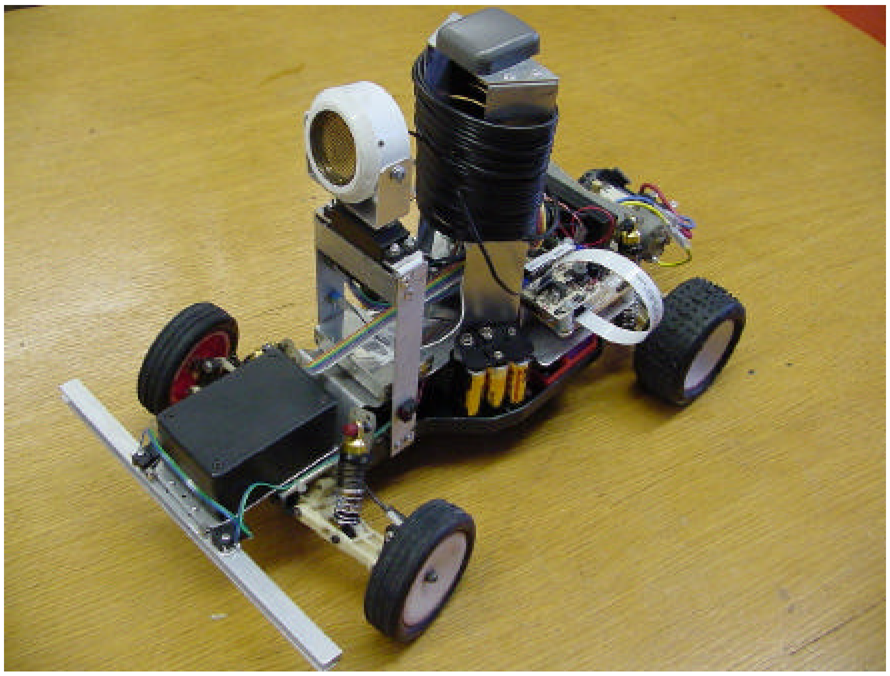


Table of Contents

Page number

Abstract:

The removal of selective availability from Global Positioning System (GPS) signals has increased accuracy for all GPS products. The Clinton administration allowed the removal of this intentionally introduced error for several reasons. Apparently, country’s enemies are no longer a threat with high accuracy GPS. As a result, the existing GPS products have become more valuable overnight and the GPS consumer market will expand due to lower priced technology. This project examines the feasibility of GPS integrated on a small autonomous vehicle. Many types of vehicles have adopted GPS for various reasons. We could certainly find many peaceful applications for such a platform. Initial thought surrounding such a vehicle, was for a military application. A small platform, which can navigate autonomously to a programmed coordinate, could deliver ordinance or spy on a given target.

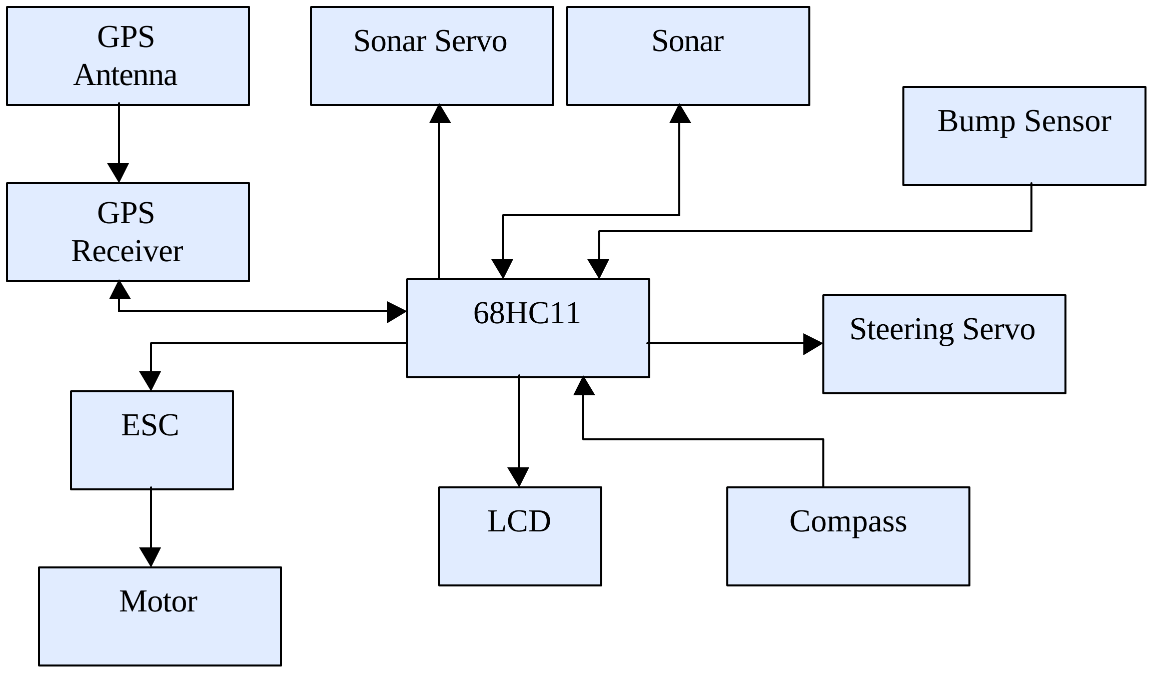
Executive Summary:

Magellan is an Associated R/C 10 car utilizing a Global Positioning System to navigate to programmed coordinates. A laptop computer is used to download Magellan’s code. An electronic speed control unit is programmed to control the R/C motor. The Ashtech G8 OEM board and antenna calculates current coordinates which are sent to the TJPro board through the SCI system. The electronic compass will also send the current heading of the vehicle via the SPI system. The 68HC11 will calculate which way it needs to track towards the given target when heading, latitude, and longitude are found. When an obstacle is detected in front of Magellan, it will slow down. If the obstacle is within 3 feet, Magellan will stop and scan the immediate area with its sonar system. Depending on the car’s orientation to the obstacle the car will back up and cut the steering to the left or right. Magellan will continue again, searching for its target coordinates until it found. When the coordinates are located the sonar will sweep the area and look for a target within the target coordinates.

Introduction:

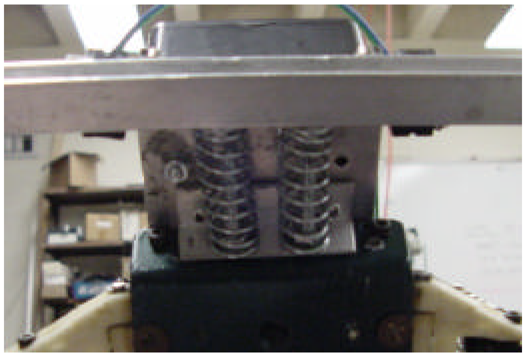
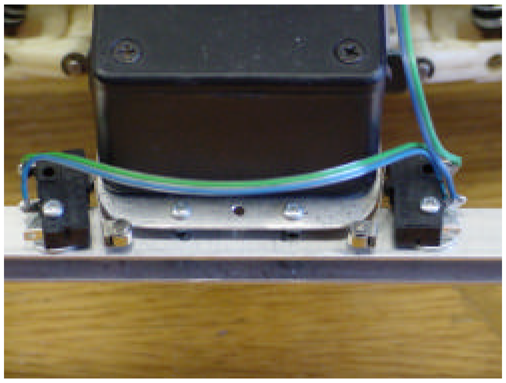
The main focus of this project is to explore the smaller and cheaper GPS units available today. The OEM boards are ideal for integration into many types of vehicles. The Ashtech receivers are among the lowest cost and least power consuming devices available. The advertised accuracy for such a unit is about 8 meters (26 ft.). The Polaroid sonar module and transducer can detect objects within 10 meters. This makes target acquisition possible for the vehicle. I did not want to spend a great deal of time designing a platform, so I chose to use an old Associated RC 10 car. Steering the vehicle would be accomplished by using an electronic compass in conjunction with the existing steering servo. The obstacle avoidance behavior would implement a sonar system located at the top of the car. As a last resort the front of the vehicle would use a bump sensor to detect any close obstacles. If time permits, I would also like to mount a LCD on the platform. The LCD could display various errors and/or goals that the user wants accomplished.

Integrated System:



Block Diagram

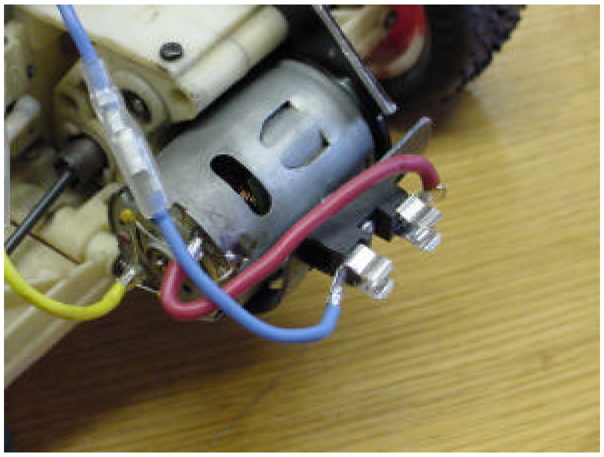
The heart of the vehicle will be a Motorola 68HC11 microcontroller. The MTJPRO11A board has all the ports and memory necessary to implement this design. After the program is loaded into the 68HC11, the laptop serial cable is disconnected and the G8 serial cable is connected in its place. This all happens using the SCI system. After the electronic compass is calibrated, data is sent from the compass to the 68HC11 via the SPI system. The motor driver is a Rooster Reversible electronic speed control (ESC) unit. After the ESC is programmed, it receives small offset PWM signals to control motor speed. The offset PWM signals are generated with the Output Compare system. The steering and sonar servos are controlled in the same manner. The sonar system utilizes the Input Capture system and an added output port. The bump sensor is connected to the A/D converter.

The car has four different power supplies, which are all Nickel Metal Hydride batteries. The motor uses the standard R/C 6-cell battery configuration. The sonar has a 6-cell battery pack. The servos, GPS receiver and antenna, ESC, and compass have a 6-cell battery supply. The GPS receiver has a 2-cell battery backup supply in order to allow the receiver to warm start.

Mobile Platform:

The platform is an old original Associated RC10 car that someone gave to me. I did have to buy various chassis parts such as axles, springs, and a shock rebuild kit. The mounting for all the subsystems is done with aluminum. The car will not have a body at all, unlike most RC vehicles. I do not want to concentrate on aesthetics; however, I spent a good deal of time to choose my sensor and circuit layout carefully. I believe this to be an important aspect of any engineering. The result was a robust and functional platform. In fact, I accidentally dropped Magellan out of my truck and there was no apparent damage to the platform. The only flaw in my design was that I did not make the motor battery easily accessible for removal. I can charge the battery in its case on the car, but if I needed to remove it I would have dismantle the entire platform. This was not a problem until demo day when my battery was low and I could not replace it with another. I did not realize how long it takes to recharge the Nickel Metal Hydride batteries. To help reduce the risk of damage from a runaway car during testing, I installed a fuse onto the motor. In case the vehicle performed unexpectedly, I could pull a string tied to the fuse and shut the motor down.



Actuation:

The vehicle uses a high torque Mabuchi RS540SH RC motor for propulsion. The high torque reduces the amount of power consumed in a typical RC car. I was also concerned with excessive weight effects. The high torque motor solves these issues. Previous students have had problems controlling their vehicles with the electronic speed controls they used. Novak e-mailed me with pertinent data specs concerning the PWM that controls the motor. Since their ESC is programmable and has a linear relationship between the PWM and speed, the Rooster Reversible seemed like the logical choice. The ESC is programmed by sending a neutral PWM to the ESC. After this happens, a button on the ESC, then a key on the keyboard is pushed. Software I have written sends the ESC a full forward PWM, a full reverse PWM, and a neutral PWM signal. The ESC is then ready for use. An offset value is added to the neutral PWM signal to move the car forward and back. It is good idea to send an additional signal for a brief moment before the offset in order to give the car a boost in correct direction. The steering and sonar servos control what direction Magellan is pointing when moving to its target.

Sensors:

The GPS system is made up of a receiver and antenna. The HC6811 sends commands to the GPS receiver using the Motorola's SCI system. The receiver has to be initialized with set commands ($PASHS) before data can be extracted for processing. When all initializations are complete, $PASHS,NME,GLL,A,ON is sent to the receiver to get the proper coordinates. The response message is of the form: $GPGLL, m1, c1, m2, c2, m3, c3*cc.

GLL Message Structure

Parameters	Description	Range
m1	Latitude component of position in degrees and decimal minutes (ddmm.mmmmmm)	0-90°
c1	Direction of latitude N = North, S = South	N or S
m2	Longitudinal component of position in degrees and decimal minutes (ddmm.mmmmmm)	0-180°
c2	Direction of longitude E = East, W = West	E or W
m3	Current UTC time of position fix in hours, minutes, and seconds (hhmmss.ss)	00-235959.50
c3	Status, A: valid, V: invalid	'A' / 'V'
*cc	Checksum	

The string of characters is put into an array and the required data fields are read from the array. The HC6811 will use this data in conjunction with the compass data to determine course track.
Preliminary data suggested inaccurate data when used in area with obstructed views of the sky. Trees surround my apartment, does not allow me to do any testing in this area. The data for my apartment position is shown in (table1, chart1-1 and chart1-2). The standard deviation is too great to be a useful sensor. I tried acquiring data on the top level of the parking garage across from Shands hospital. The data is shown in (table2, chart2-1 and chart2-2). The accuracy is far greater at the garage than my apartment. Since the area of demo will be relatively small enough, I only need to be concerned with the second's part of the latitude and longitude. After the compass is initialized, the 68HC11 polls the compass for data when needed. The data is sent serially through the SPI system. The sonar system requires a signal to be sent to the transducer to begin a measurement of distance. A signal is sent out to the module and pin PA2

at the same time. When an echo is returned, pin PA1 goes high and the time is measured. This eventually allows a distance to be calculated.

Behaviors:

The most interesting behavior may be the ability to find a set of preprogrammed coordinates. Magellan begins by determining if it is at its target coordinates or elsewhere. If it is not where it should be, current latitude, longitude and heading will help to steer the car in the correct direction (flowchart 1). Once it is moving the sonar constantly checks for obstacles in its way. If there is something within 5 to 15 feet, the car will slow down. If there is something within 3 feet in front of Magellan, the vehicle will stop, scan the vicinity and determine the best route around the obstacle. A hard target location could be acquired within the target coordinates. When inside the target perimeter, the sonar looks for a hard target to lock onto. Magellan can store several coordinates into the computer and let the car trace a course through each waypoint. I will also want the vehicle to be able to steer a straight course.

Experimental Layout and Results:

All experiments were conducted with a fully operational platform. The sonar worked perfectly until it was installed on the car. The height of the transducer caused premature echo returns. This was probably due to the proximity of the ground. I could get relatively good distance measurements with the transducer pointed at a 25° from the horizon. For added accuracy I made three distance measurements, took the average, and used the average distance for my avoidance routine. The vehicle must also operate on a smooth surface or the sonar will give false readings. The compass never performed as expected. I received data from the serial port, but it looked like erroneous data. The GPS system was initially connected to the laptop. I tested accuracy by keeping the antenna in one location and receive several samples of data. By utilizing the battery backup I could power up the receiver with a warm start. I did not have to wait several minutes each time I wanted to receive coordinates. I could easily extract the latitude and longitude from the data string and put them into an array.

Conclusion:

The most significant problem with this project was the compass. For some reason I could not get good data from the compass. I'm still trying to determine whether it is my software or the compass hardware that is the problem. The ability to get a current heading is critical for producing any algorithm, which controls the car's movements. I can get a heading from the GPS unit using the ($PASHS,NME,POS) command. I have reservations about the practicality of using this data because there is a certain amount of error when calculating the coordinates. I could get by without this data, but tends to make the platform's motion jerky. The sonar and GPS worked perfectly for the most part. I thought the GPS would be the most difficult part of this project, instead it was the easiest to implement. I may want to include a differential global positioning system (DGPS) unit on my platform at a later date. Although I did not have any of the control issues with my ESC, it would probably be in my best interest to implement a velocity feedback into the ESC for safety reasons. Fortunately the GPS has the ability to measure horizontal and vertical velocities. I could parse that data for a feedback system later.

References:

Joseph Jones, Bruce Seiger & Anita Flynn, *Mobile Robots: Inspiration to Implementation*, 2nd edition, A.K. Peters Publishers, Natick, MA, 1998

Mekatronix – http://www.mekatronix.com

Magellan - http://www.magellangps.com

Precision Navigation - http://www.precisionnavigation.com

Appendix A:

```
/*************************************************************************
*                    Magellan program                                    *
*                    Date: 4/12/2001                                     *
*                    Name: Nicholas Ivano                                *
*                    Version 1.0.0                                       *
* This program will allow Magellan to autonomously navigate to a set of  *
* pre-programmed coordinates. Magellan will use sonar to detect any      *
* obstacles.                                                             *
*************************************************************************/

/***************************** INCLUDES *********************************/
#include <nickbase.h>
#include <stdio.h>
#include <mil.h>
#include <nickhc11.h>
/************************* END OF INCLUDES ******************************/

/****************************** DEFINES *********************************/
#define false 0
#define true 1
#define bumper analog(0)
// DEFINES FOR COMPASS.
#define reset 0x7
#define calibrate 0x6

#define steer_right 2800
#define steer_right40 2600
#define steer_right20 2400
#define steer_straight 1980
#define steer_left20 1650
#define steer_left40 1250
#define steer_left 1080

#define sonar_20 1070
#define sonar_50 1700
#define sonar_90 2500
#define sonar_130 3300
#define sonar_160 4000
// THESE VALUES ARE THE PULSEWIDTHS.
#define full_forward 4000
#define full_reverse 1000
// THESE VALUES ARE ADDED TO THE EXISTING NEUTRAL PULSEWIDTHS.
#define forward0 1
#define neutral 0
#define reverse0 -1
#define forward1 150
#define forward2 200
#define reverse -400

#define for_boost 700
#define rev_boost -900

#define reset0 0x3
#define reset1 0x7
#define calibrate0 0x6
#define calibrate1 0x7
```

```
#define poll0 0x5
#define poll1 0x7
/************************** END OF DEFINES *****************************/

#pragma interrupt_handler SPI_isr;

/**************************** PROTOTYPES ********************************/
void SPI_isr(void);
void ouch(void);
void avoid(void);
void track(void);
void arbitrate(void);
void move(int , int);
void reset_flags(void);
void init_esc(void);
void init_compass(void);
void init_gps(void);
void init_spi(void);
void cal_compass(void);
void sonar_ping(void);
float sonar_range(void);
void sonar_scan(void);
void avoid_routine(void);
int target(float , float);
int compute_head(int , float , float);
void esc(int);
float get_head(char *string , int);
float decode_output(char *string , int);
void store_message(void);
void store_message2(void);
void init_sci(void); /* Set up the SCI port  */
/************************ END OF PROTOTYPES ****************************/

/************************* GLOBAL VARIABLES ****************************/
int ouch_flag, avoid_flag, track_flag;
int o_velocity, a_velocity, t_velocity;
int heading, speed, steer;
float distance, longitude, latitude;

char t = '0';
int good_neutral = false;               // FLAG SET WHEN IN NEUTRAL POSITION.
int fivefeet = false;                           // FLAG SET WHEN SONAR DETECTS WITHIN 5 FEET.
int fms, rms, dfms, drms;               // MOTOR SPEED VARIABLES.
char YoN, prompt;                                       // MENU VARIABLES.
int PW, FPW, RPW, NPW;                                  // PULSEWIDTH VARIABLES.
float distances[5] = {0};               // SCANED DISTANCES ARE PUT INTO AN ARRAY.
float ave_dist[3] = {0};
float fl_distances[4] = {0};            // FORWARD_LOOK DISTANCES ARE PUT INTO AN ARRAY.

int count = 0;
int user_longitude;
int user_latitude;
char ch;
float power;
int cmd_rdy = 0;
int lat_rdy = 0;
int long_rdy = 0;
char message[125] = {0};
char gga[] = "$PASHS,NME,GGA,A,OFF\r\n";
char vtg[] = "$PASHS,NME,VTG,A,OFF\r\n";
char coords[] = "$PASHS,NME,GLL,A,ON\r\n";
char position[] = "$PASHS,NME,POS,A,ON\r\n";
char targcord[] = "2937.257320,N,08221.457220,W";
char testcord[] = "$PASHR,POS,0,06,214619.50,2937.259760,N,08221.450330,W,"
                                                    "00043.110,1234,331.0,000.7,000.0,02.7,01.2,02.4,01.6,HC00*6C";
int r, i;
/*********************** END OF GLOBAL VARIABLES ************************/

void init_sci(void){
  CLEAR_BIT(SPCR,0x20);
  BAUD = 0x31;  /* 0xbO is 9600, 0x35 is 300 baud, 0x31 is 4800 baud*/
  SCCR2 = 0x0C;
  CLEAR_BIT(SCCR1, 0x10);                    // 8 data bits, and 1 stop bit.
}
/************************ End init_serial ****************************/

/************************* MAIN FUNCTIONS ****************************/
// FUNCTION DETECTS WHEN BUMPER STRIKE.
void ouch(void){
        if(bumper > 50){
                ouch_flag = 1;
                o_velocity = forward0;
        }
        else
                ouch_flag = 0;
}

// FUNCTION CONTAINS AVOIDANCE PROCEDURES.
void avoid(void){
        int i = 0;
        while(i < 3){
                sonar_ping();
                distance = sonar_range();
                ave_dist[i] = distance;
                wait(100);
                i++;
        }
        distance = (ave_dist[0]+ave_dist[1]+ave_dist[2])/3;

        if(distance < 3 && distance >= 0.5){
                avoid_flag = 2;
        }
        else if(distance >= 3 && distance <= 5){
                avoid_flag = 1;
                a_velocity = forward1;
        }
        else if(distance > 5 && distance <= 15){
                avoid_flag = 1;
                a_velocity = forward2;
        }
        else {
                avoid_flag = 0;
                a_velocity = forward2;
        }
}
```

```
// FUNCTION CONTAINS GROUND TRACK PROCEDURES.
void track(void){
		int target_found;
		heading = get_head(message , 70);
		longitude = decode_output(message , 46);
		latitude = decode_output(message , 31);
		target_found = target(longitude, latitude);
		if(target_found == 1){
				track_flag = 1;
				t_velocity = forward0;
		}
		else {
				track_flag = 2;
				t_velocity = forward2;
		}
}

// FUNCTION CONTAINS ARBITRATION PROCEDURES.
void arbitrate(void){
		if(ouch_flag == 1){
				speed = o_velocity;
				steer = steer_straight;
		}
		else if(avoid_flag != 0){
				if(avoid_flag == 2){
						avoid_routine();
				}
				else {
						speed = a_velocity;
						steer = steer_straight;
				}
		}

		else if(track_flag == 2){
						speed = t_velocity;
						steer = compute_head(heading, longitude, latitude);
//						steer = steer_straight;
		//DEBUG.
				}
		else {
				speed = forward0;
				steer = steer_straight;
		}
}

// FUNCTION CONTAINS MOVEMENT PROCEDURES.
void move(int sd, int st){
		if(ouch_flag == 1){
				servo(1, st);
				esc(sd);
				esc(rev_boost);
				esc(reverse);
				servo(1, steer_left20);
				wait(3000);
				servo(1, steer_left);
				wait(200);
				esc(0);
```

```
                wait(200);
                servo(1, steer_straight);
                speed = forward2;
                steer = steer_straight;
        }
        else if(avoid_flag == 1){
                servo(1, st);
                esc(sd);
        }
        else if(track_flag == 2){
                servo(1, st);
                esc(sd);
        }
        else {
                servo(1, steer_straight);                                   //DEBUG.(st should be straight)
                esc(forward2);
        }
}

// FUNCTION RESETS ALL PRIORITY FLAGS.
void reset_flags(void){
        ouch_flag = 0;
        avoid_flag = 0;
        track_flag = 0;
}
/********************** END OF MAIN FUNCTIONS *************************/

float get_head(char strng[125], int index){
        float number, digit;
        number = 0;
        for(i = index; i < (index + 3); ++i){
                ch = strng[i];
                ch = ch - 48;
                digit = (float) ch;
                if(digit < 0 || digit > 9)              // CHECK FOR VALID DIGITS.
                        return -1;
                if(i == index)
                        number += digit * 100;
                else if(i  == index + 1)
                        number += digit * 10;
                else
                        number += digit;
        }
                return number;
}
float decode_output(char strng[125], int index){
        float number, digit;
        number = 0;
        for(i = index; i < (index + 5); ++i){
                ch = strng[i];
                ch = ch - 48;
                digit = (float) ch;
                if(digit < 0 || digit > 9)              // CHECK FOR VALID DIGITS.
                        return -1;
                if(i == index)
                        number += digit * 10000;
                else if(i == index + 1)
                        number += digit * 1000;
                else if(i == index + 2)
                        number += digit * 100;
                else if(i == index + 3)
                        number += digit * 10;
                else
                        number += digit;
        }
        return number;
}

void store_message(void){
        do {
                ch = get_char();
                } while(ch != '$');
                for(i = 0; ch != '*'; ++i){
                        message[i] = ch;
                ch = get_char();
                }
}

// INITIALIZES THE ELECTRONIC SPEED CONTROL.
void init_esc(void){
        //TEST TO SEE IF MOTOR IS IN NEUTRAL POSITION.
        while(good_neutral == false){
                printf("Enter motor neutral position (2700<PW<2800):   ");
                NPW = read_int();

                servo(0, NPW);
                printf("Is motor in neutral position (Y/N)?:\n");
                YoN = getchar();
                if(YoN == 'Y'|| YoN == 'y')
                        good_neutral = true;
        }

        //TEST TO SEE IF READY TO PROGRAM ESC.
        YoN = 'N';
        while(1){
                printf("\nHit 'y' key when 'set' pushed on ESC.\n");
                YoN = getchar();
                if (YoN == 'y' || YoN == 'Y')
                        break ;
        }

        //SET FULL-FORWARD THROTTLE THEN WAIT FOR 2 SECONDS.
        PW = full_forward;
        servo(0, PW);
        wait(2000);

        //SET FULL-REVERSE THROTTLE THEN WAIT FOR 2 SECONDS.
        PW = full_reverse;
        servo(0, PW);
        wait(2000);

        //SET NEUTRAL THROTTLE THEN WAIT FOR 2 SECONDS.
        servo(0, NPW);
        wait(2000);
}
```

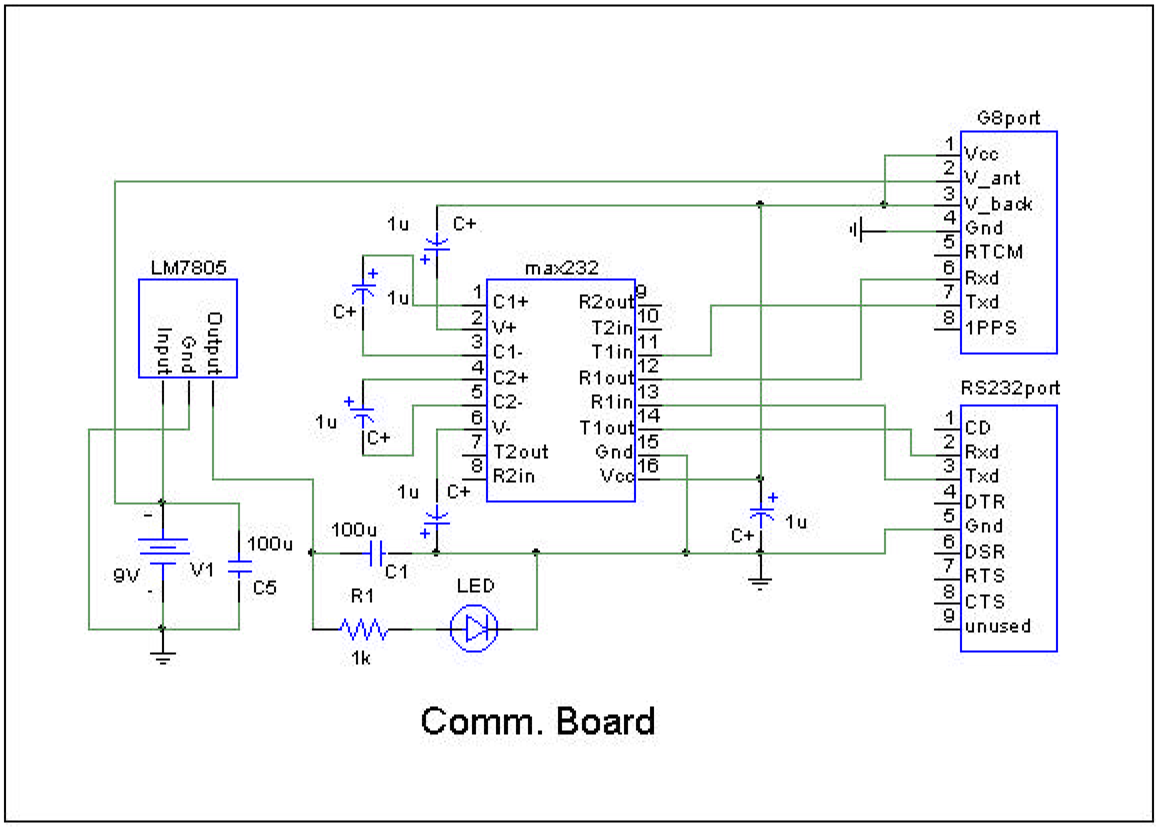
```
// INITIALIZES THE SONAR SYSTEM.
void init_sonar(void){
            INTR_OFF();                             // TURN OFF INTERRUPTS.
            SET_BIT(TCTL2, 0x14);                   /* CAPTURE RISING EDGE */
            CLEAR_BIT(TCTL2, 0x28);                 /* FOR IC1 & IC2. */
            INTR_ON();                              // TURN ON INTERRUPTS.
}

void init_gps(void){
            write(gga);                     // SHUTS OFF $GPGGA MESSAGE.
            write(vtg);                     // SHUTS OFF $GPVTG MESSAGE.
            write(position);    // TURNS ON $PASHR MESSAGE.
            wait(1000);
}

void init_spi(void){
            INTR_OFF();                                     // TURN OFF INTERRUPTS.
            SET_BIT(SPCR, 0xC0);            // TURN ON AND ENABLE SPI INTERRUPT.
            CLEAR_BIT(SPCR, 0x10);                  // SLAVE MODE.
            SET_BIT(SPCR, 0x0C);            // CPOL = 1, CPHA = 1.
            INTR_ON();                              // TURN ON INTERRUPTS.
}

void init_compass(void){
            PORTOUT = reset1;                       // HOLDS RESET, P/C AND CAL HIGH.
            wait(20);                                       // WAIT 20 MILLISECONDS.
            PORTOUT = reset0;                       // HOLDS RESET LOW, P/C AND CAL HIGH.
            wait(20);                                       // WAIT 20 MILLISECONDS.
            PORTOUT = reset1;                       // HOLDS RESET, P/C AND CAL HIGH.
            wait(600);                                      // WAIT 20 MILLISECONDS.
            cal_compass();
            wait(100);
}

void cal_compass(void){
            PORTOUT = calibrate0;           // HOLDS CAL LOW, P/C AND RESET HIGH.
            wait(20);                                       // WAIT 20ms.
            PORTOUT = calibrate1;           // HOLDS CAL, P/C AND RESET HIGH.
            wait(20);                                       // WAIT 20ms.
            PORTOUT = calibrate0;           // HOLDS CAL LOW, P/C AND RESET HIGH.
            wait(20);                                       // WAIT 20ms.
            PORTOUT = calibrate1;           // HOLDS CAL, P/C AND RESET HIGH.
            wait(50);
            while(t != 'y'){
                        printf("Turn vehicle  180 degrees and hit 'y' key.\n");
                        t = getchar();
            }
            PORTOUT = calibrate0;           // HOLDS CAL LOW, P/C AND RESET HIGH.
            wait(20);                                       // WAIT 20ms.
            PORTOUT = calibrate1;           // HOLDS CAL, P/C AND RESET HIGH.
            wait(20);                                       // WAIT 20ms.
}

// SETS THE PROPER STEERING ANGLE FOR PARTICULAR OBSTACLE.
void avoid_routine(void){
            servo(1, steer_straight);
            esc(forward0);
            sonar_scan();
```

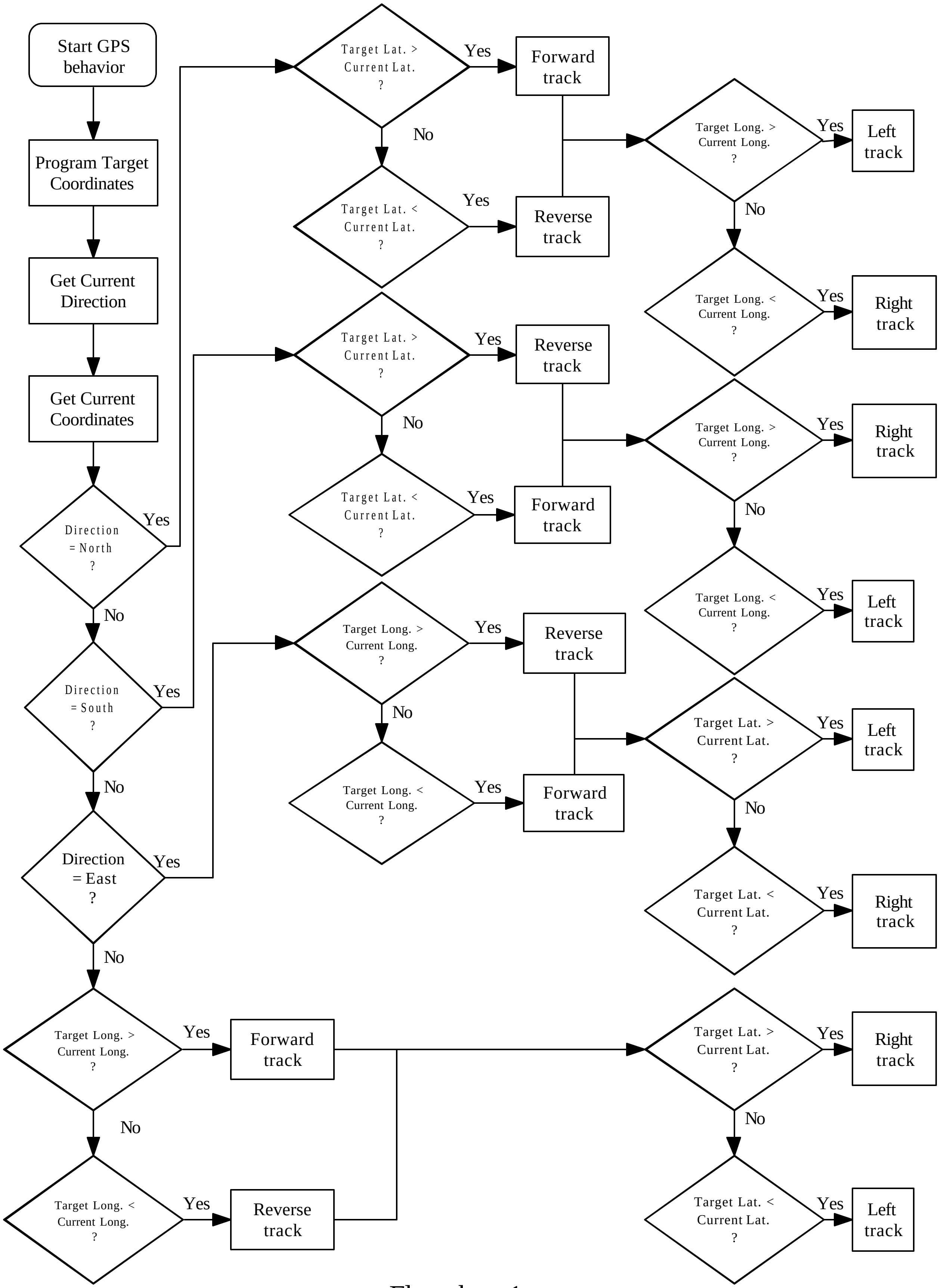
```
	if(((distances[3] < distances[2]) || (distances[4] < distances[2]))
		&& ((distances[1] > distances[3]) || (distances[0] > distances[4]))){
		esc(rev_boost);
		esc(reverse);
		wait(1000);
		servo(1, steer_left20);
		wait(1500);
		servo(1, steer_left40);
		wait(300);
		esc(0);
		wait(1000);
		servo(1, steer_straight);
		wait(200);
		speed = forward2;
		steer = steer_straight;
	}
	else if(((distances[1] < distances[2]) || (distances[0] < distances[2]))
				&& ((distances[3] > distances[1]) || (distances[4] > distances[0]))){
		esc(rev_boost);
		esc(reverse);
		wait(1000);
		servo(1, steer_right20);
		wait(1500);
		servo(1, steer_right40);
		wait(300);
		esc(0);
		wait(1000);
		servo(1, steer_straight);
		wait(200);
		speed = forward2;
		steer = steer_straight;
	}
	else {
		esc(rev_boost);
		esc(reverse);
		servo(1, steer_left20);
		wait(1500);
		servo(1, steer_left);
		wait(200);
		esc(0);
		wait(200);
		servo(1, steer_straight);
		speed = forward2;
		steer = steer_straight;
	}

	avoid_flag = 1;				// SETS TO 1 FOR move().
}
int target(float lat, float lon){
	float lonerror;
	float laterror;
	float longit, latit;
	longit = decode_output(targcord, 20);
	latit = decode_output(targcord, 5);
	lonerror = longit * 0.00001;
	laterror = latit * 0.00001;
	if((lon <= (longit + lonerror)) && (lon >= (longit - lonerror))
                && (lat <= (latit + laterror)) && (lat >= (latit - laterror)))
                return 1;
        else
                return 0;
}

int compute_head(int head, float lon, float lat){
        float target_lat, current_lat1, current_lat2, current_lat3;
        current_lat1 = lat;
        if(target_lat > current_lat1){
                current_lat2 = decode_output(message, 31);
                wait(500);
                if(target_lat > current_lat2){
                        if(current_lat1 < current_lat2)
                                steer = steer_straight;
                        else
                                steer = steer_left20;
                        current_lat1 = current_lat2;
                }
                else {
                        current_lat3 = decode_output(message, 31);
                        wait(500);
                        if(target_lat < current_lat3){
                                if(current_lat2 > current_lat3)
                                        steer = steer_straight;
                                else
                                        steer = steer_right20;
                                current_lat1 = current_lat3;
                        }
                }
        }
        else {
                steer = steer_straight;
                current_lat1 = current_lat3;
        }
        return steer;
}

// SETS THE DESIRED MOTOR SPEED WITH INITIAL BOOST.
void esc(int pulsewidth){

        if (pulsewidth == 0){                           // SETS MOTOR TO ZERO SPEED.
                servo(0, 0);
        }
        else if (pulsewidth == -1){             // SETS MOTOR TO ZERO SPEED WHEN MOVING BACK.
                servo(0, NPW + 700);
                wait(200);
                servo(0, 0);
        }
        else if (pulsewidth == 1){              // SETS MOTOR TO ZERO SPEED WHEN MOVING FROWARD.
                servo(0, NPW - 900);
                wait(200);
                servo(0, 0);
        }
        else if (pulsewidth > 0){               // SETS MOTOR TO FORWARD SPEED.
                servo(0, NPW + pulsewidth);
        }
        else {
```

```
			servo(0, NPW + pulsewidth);
		}
}

// SENDS OUT A PING.
void sonar_ping(void){
		TFLG1 = 0xFF;					// CLEARS ALL FLAGS.
		SET_BIT(PORTOUT, 0x80);				// SETS BIT 7 AT OUTPUT PORT.
		wait(30);						// WAITS 30 MILLISECONDS.
		CLEAR_BIT(PORTOUT, 0x80);			// CLEARS BIT 7 AT OUTPUT PORT.
}

// FINDS THE DISTANCE FROM CAR TO OBSTACLE.
float sonar_range(void){
		if(((TFLG1) & 0x02) == 0)
			return -1.0;						// Did not capture echo.
		else if((((TIC2 - TIC1) >> 1) * 0.000569) < 0)
			return -(((TIC2 - TIC1) >> 1) * 0.000520);	// Captured echo.
		else
			return (((TIC2 - TIC1) >> 1) * 0.000520);	// Captured echo.
}

// SCANS 20-160 DEGREES AND PUTS DISTANCES INTO AN ARRAY.
void sonar_scan(void){
		float distance;
		int i;
		for(i = 0; i < 5; ++i){
			switch(i){
				case 4: servo(2, sonar_160);		// TURNS SONAR 160 DEGREES.
						break;
				case 3: servo(2, sonar_130);		// TURNS SONAR 130 DEGREES.
						break;
				case 2: servo(2, sonar_90);		// POINTS SONAR STRAIGHT.
						break;
				case 1: servo(2, sonar_50);		// TURNS SONAR 50 DEGREES.
						break;
				case 0: servo(2, sonar_20);		// TURNS SONAR 20 DEGREES.
						break;
			}
			wait(1000);
			sonar_ping();
			distance = sonar_range();
			distances[i] = distance;
			wait(1000);
		}
		servo(2, sonar_90);				// POINTS SONAR STRAIGHT.
}

/************************** END OF FUNCTIONS ***************************/

/********************************** ISR **********************************/
void SPI_isr(){					// NOT FINISHED.
   heading = SPSR;
		heading = SPDR;
}
/****************************** END OF ISR ******************************/

/****************************** MAIN **********************************/
```

```
void main(void){


	// VT100 CLEAR SCREEN.
	char c1, clear[]= "\x1b\x5B\x32\x4A\x04";

	/* VT100 position cursor at (x,y) = (3,12) command is "\x1b[3;12H"*/
	char place[]= "\x1b[1;1H";/*Home*/

	init_servotjp();
	init_sci();
	init_clocktjp();

	printf("%s", clear);
	printf("%s", place);

	printf("\tTitle\t\tmagellan.c\n"
	"\tProgrammer\tNicholas Ivano\n"
	"\tDate\t\tApril 12, 2001\n"
	"\tVersion\t\t1.0.0\n\n");

	init_analog();
	init_sonar();
	init_esc();

	printf("Enter the latitude (NORTH): ");
		user_longitude = read_int();
	printf("\nEnter the longitude (WEST): ");
		user_latitude = read_int();

	printf("\nPress bumper when ready.\n");

	START;

	init_gps();

	wait(1500);
	servo(2, sonar_90);					// SONAR LOOKS FORWARD.
	wait(2000);
	servo(1, steer_straight);
	wait(1000);
	esc(for_boost);
	esc(forward2);

	while(1){

		reset_flags();
		ouch();
		avoid();
		track();
		arbitrate();
		move(speed, steer);
	}
}
/***************************** END OF MAIN *****************************/
```

Appendix B:


G8port
1 Vcc
2 V_ant
3 V_back
4 Gnd
5 RTCM
6 Rxd
7 Txd
8 1PPS
RS232port
1 CD
2 Rxd
3 Txd
4 DTR
5 Gnd
6 DSR
7 RTS
8 CTS
9 unused
max232
1 C1+
2 V+
3 C1-
4 C2+
5 C2-
6 V-
7 T2out
8 R2in
9 R2out
10 T2in
11 T1in
12 R1out
13 R1in
14 T1out
15 Gnd
16 Vcc
LM7805
Input
Gnd
Output
1u
C+
100u
C1
C5
9V
V1
R1
1k
LED
Comm. Board


Start GPS behavior
Program Target Coordinates
Get Current Direction
Get Current Coordinates
Direction = North ?
Direction = South ?
Direction = East ?
Target Lat. > Current Lat. ?
Target Lat. < Current Lat. ?
Target Long. > Current Long. ?
Target Long. < Current Long. ?
Forward track
Reverse track
Left track
Right track
Yes
No

Flowchart 1