Navigator

Designer: Jeff Brigman
Intelligent Machine Design Lab
Final Report
Summer 1999

Table of Contents

Abstract

Navigator is planned as an experiment in auto-navigation. Build on a large R/C platform, Navigator will use Global Positioning System (GPS) satellites to track its position and navigate towards a given destination. In doing so, it will challenge the accuracy of various GPS schemes, and provide insight into the feasibility of self-navigating vehicles for real world use.

Executive Summary

Navigator started off as an ambitious project to see just how well a robot could handle navigation. With a GPS and compass for direction finding, and sonar for eyesight Navigator has all of the tools necessary to accomplish just such a task. A 1/10 scale R/C platform provides the necessary mobility, and 3 NiCd battery packs supply the necessary power.

In building Navigator, several things became evident. Things that seem simple or even trivial can take hours to figure out when they don't work like they're supposed to. Also, you can never get the hardware finished early enough. It would probably take another month to finish the code to the extent I would like to. At present time, the GPS communication interface is not working properly, although the other systems are functioning and fully integrated.

One important thing to mention is that the differential receiver for the GPS was omitted during the course of the project due to cost and time constraints. That will be something added to Navigator in the future as I continue to work with it.

Introduction

My interest in vehicle self-navigation is based on navigation systems for other means of transportation. For many years auto-helms have been available for boats. You simply pick a heading and let the boat steer itself. Similarly, planes have auto-pilots. So why not do the same for a land-based vehicle? The main reason is the number of obstacles present. There aren't many obstacles on the open water or in the air for the navigation system to deal with. However, on land there are all sorts of obstacles to be dealt with: buildings, pedestrians, other vehicles, etc.

To handle large scale navigation, the Global Positioning System (GPS) will be used. GPS is a system of 24 satellites which transmit position signals to earth. A GPS receiver uses data from 3 to 12 of these satellites to figure its exact location. Due to atmospheric conditions and other error factors, normal accuracy is somewhere around 25 meters. However, the Department of Defense, who runs the satellites, feels it is in their best interest if we do not have a precise locating system, so they skew the data provided by the satellites so that error may be as much as 100 meters. This limitation makes precise navigation nearly impossible..

Fortunately, there is another technology known as Differential GPS (DGPS) which allows much better accuracy. It essentially uses two GPS receivers, one mobile, the other a base station. The base station is placed at a location whose precise coordinates are known. It compares its known position to that given by its GPS receiver. The difference

(or offset) is then broadcast on an radio frequency. The mobile GPS receives this offset data and uses it to correct any errors there may be. This method provides highly accurate positioning data. From one source I have read, differential GPS accuracy is in the 1-10 meter range, more specifically with accuracy within 2 meters 65% of the time. Whether or not these claims are true can only be determined through experimentation.

Integrated System

The brains for this robot will be a Motorola 68HC11 micro-controller. It is fitted with a Mekatronics ME11 daughter board, which provides an additional 32K of RAM and extra output ports, as well as memory backup and voltage regulation. Each external sensor will communicate directly with the 68HC11. Using data from a GPS, compass, sonar, and bump switches, it will arbitrate between a set of behaviors. The 68HC11 must provide signals to drive the motor and the two servos.

Mobile Platform

Due to the large scale use of GPS and the need to receive satellite signals, this robot must operate outdoors. As such a robot body is needed that is robust enough to move around in almost any terrain. Also it must be large enough to hold all of the components that will be involved. To meet this requirement I selected a 1/10 scale R/C vehicle, the Kyosho Tracker 2. The vehicle is rear wheel drive using a single motor. At nearly 18 inches in length, and having 4.5 inch tires, it is large enough to hold everything, and able to crawl over almost any terrain. It also has a powerful DC motor, which is important when traveling outdoors. It is also capable of traveling a large distance in a relatively short amount of time, which will be important when it is time to demonstrate its navigation abilities.

The 68HC11 is mounted underneath the truck body, in a protective box designed to protect it from dirt and other hazards. The compass is placed up front, between the front shock towers, to keep it as far from the motor as possible. The sonar unit is mounted on top of the truck cab, as far off the ground as possible. This should minimize false readings from ground reflection.

A total of 3 7.2 volt battery packs will be used on Navigator. Two 1500mAh batteries are connected in parallel to provide extended battery life for the motor. A third battery, this one 1700mAh, is used to power the micro-controller and most of the other electronics. This multi-battery arrangement lets me isolate the electronics from the

voltage spikes that are sure to occur with the use of the DC motor. The GPS runs off of 4 AA batteries contained within the unit. A pair of 9V alkalines are connected in series and regulated to 12 volts to power the sonar unit.

Actuation

Basic movement is provided by the DC motor that was provided with the R/C kit. A standard servo is used to control the steering at the front wheels. The sonar is mounted on top of a second servo so that it can sweep back and forth. The idea here is to allow the sonar to detect objects that are not directly in the robot's path.

Originally I built and planned to use a motor driver board using 75 amp MOSFETs. After a few programming difficulties and seeing the problems other students were having with home-built motor driver boards, I opted to buy an Electronic Speed Control specifically designed for the R/C motor I am using. It operates off a standard servo input, converting that into a PWM signal to drive the motor at the desired speed. This makes programming a bit easier because I only need to run a servo program instead of mixing one with the PWM generating motor program.

Sensors

Bump Switches

To provide basic collision detection, bump switches are connected to the front bumper to detect when an object is hit. This is an absolute last resort, as the vehicle travels fairly fast, and hard impacts will be avoided at all cost. Since my PortA pis are being used by output compares (servos, motor) and PortD is tied up by the SPI and SCI functions, I decided to use an analog port, even though this is a digital signal. An analog reading of 255 indicates no contact, while 0 indicates that a collision has occurred.

Sonar

Since the robot will be running outdoors, light levels will be too high for IR to be of any use. Also, the useful distance of IR is less than 2 feet, which would be inadequate for the speeds at which the robot will be traveling. So, sonar has been chosen for object detection. I obtained a SonaSwitch Mini-S sonar unit from Scott Jantz which takes an input voltage of 8-16V and returns an analog value corresponding to the distance to an object. As currently calibrated the value ranges from 4.96 V at 18ft to .03V at around 6 inches. The sonar is mounted on top of a servo, which gives it the ability to look in different directions. Since sonar relies on line-of-sight the sonar is mounted in the open, on top of the truck cab. This raises it about 8 inches off the ground and should minimize ground reflection and false readings.

Electronic Compass

While a GPS can be used to give a direction of travel, it is only useful when traveling at a high rate of speed. It is important for the robot to know what direction it is pointing at all times, whether moving or not. The compass I used is the Precision Navigation Vector 2XG. Since this is a gimbaled model it allows the compass to function even if the robot is tilted somewhat. The compass is interfaced with the 68HC11 through the SPI system. When polled, the 2XG returns the current heading in binary form. To minimize the possibility of interference, the compass is mounted as far forwards as possible in the robot, since the motor is at the rear.

GPS receiver

The most important sensor for this robot, and the most challenging to get working, is the GPS receiver. For this robot I selected the Garmin GPS 12. The GPS 12 can communicate with other devices using the NMEA 0183 format. NMEA 0183 is an interface specification and protocol standard provided by the National Marine Electronics Association. NMEA 0183 is loosely based on RS-232 at 9600 Baud with 8,N,1. However, what should be a +-12V signal is only 7V, and then only the positive voltage is broadcast (negative is simply 0V). In testing, the computer serial port's RS-232 converter was able to make sense of this, but the one on the 68HC11 EVBU was not. To make this work, I built my own circuit to decode the signal into readable format. Using a simple voltage divider network I drop the voltage from 7 to 5V. Since RS-232 data is also inverted, I then run the signal through a 74'04 hex inverter. The output from that chip is

taken directly into the 68HC11 serial data receive pin., bypassing the EVBU's RS-232 converter altogether.

Behaviors

Object Avoidance

Primarily, sonar isused for object avoidance. Generally speaking, the sonar will face forward and look for objects. When something is detected it will look to the left and right to determine which direction is the best avoidance route. If for some reason the sonar fails to see an object, the bumper mounted switches will detect a collision and the robot will back up and turn to either direction to go around the object.

Wall Following (for large objects)

If a large object, say a building is encountered, the robot should not just randomly turn back towards the object, but rather travel to the end of it and then readjust its heading. To do this, the sonar platform will alternate checking for objects in front of the robot, and measuring the robot's distance from the wall. Steering adjustments will be made as necessary to follow the wall, or avoid other objects. Once the wall is cleared, the robot will use the GPS to get a new bearing, the compass to turn to that heading, and continue navigating from that point on. A status flag will be maintained for the robot to know at all times if an object is being followed.

Navigating to destination point

This is the main behavior for the robot. The GPS is programmed with the coordinates for a specific location. It provides a compass heading to get to the destination point, and the distance of the robot from that destination point. The robot will use the compass to steer to the given compass heading and follow along that path. If another behavior takes over (object avoidance or wall following), the navigation behavior will eventually regain control. At that point it will query the GPS for a heading, and steer in that direction. Because of the GPS, the robot will never become lost. It will always be able to get a compass heading that will take it to the destination point.

Detect destination object

Since GPS does not have perfect accuracy, the robot may not be able to reach the exact destination point, but it should be near the destination. So, I plan to place some object at the destination point (which will have to be an open area). Once the GPS says the robot has reached its destination, the sonar will be used to find some object. If the object is detected, its heading will be estimated by comparing the sonar servo position to the electronic compass heading. The robot will steer to that heading and adjust as needed until it reaches the object. If the object is not immediately seen, the robot will search in that area until the sonar reads some object. Then it will head towards the object as stated above.

Experimental Layout and Results

Sonar

The instructions for calibrating the sonar unit were nearly worthless, as they didn't really give any useful information. By simply setting the unit about 20' from a wall and dialing the potentiometers the analog value returned was 4.96 (the maximum), I achieved excellent range in less than 10 minutes. Once dialing this in, I found that it had a fairly linear response.

Distance	Analog Value
6"	.08 V
2'	.46 V
4'	.83 V
6'	1.37 V
8'	1.89 V
10'	2.42 V
12'	2.97 V
14'	3.60 V
16'	4.25 V
18'	4.56 V
20'	4.96 V (max)

Compass

The Vector 2XG surprised me a little bit. Although it claims to self-calibrate, I found that the heading provided by the Vector differed from my hand-held compass. However, it seems that the offset is constant, so I simply added a line in my code to align the two values. By measuring several points I found the offset to be roughly 45-50 degrees, so I

simply added the statement “heading = heading - offset”, where offset is specified by a #define statement at the beginning of the program.

GPS

The GPS has proven to be the most difficult part of the project. It took quite some time to figure out how the (not quite) RS-232 data was encapsulated, and designing the decoder circuit for it. I was always able to get characters, but they were extended ASCII characters, nothing that should have been transmitted. Thinking there was a bit sync error between the two devices, I replaced the ceramic oscillator on the EVBU with a 8.000 MHz crystal. Only after that did we realize what sort of signal was coming out of the GPS. Even knowing and correcting the signal, a few strange things still exist. Whenever the GPS is connected to the 68HC11 receive pin, the data is somehow re-transmitted back out to the computer through the serial cable, although the code does not request that.

Conclusion

When all of the hardware is installed and the code sufficiently built up, Navigator should be able to function as a fully autonomous vehicle, interacting with any objects it encounters. Unlike navigation robots made in the past, the use of GPS can ensure that Navigator always knows where it is, and how to get to its intended destination.

However, as with all of these robots, the work is never done. At this time I have not been able to get the GPS communications code working properly. Since this was the main goal for the robot, it is a bit of a disappointment. In the spirit of the project, the robot is programmed to track towards a specific compass direction, which can be changed whenever the program is compiled. This allows the robot to illustrate it's navigation abilities even though the GPS is not able to provide constantly updated information.

Parts List

ME11 board	$20	Scott Jantz (Mekatronics)
Mini-S sonar	$50	Scott Jantz (Mekatronics)
Garmin GPS 12 Computer Interface Cable	$140	James Associates www.sni.net/~lwjames
Garmin Interface Cable Ends (2)	$15	Wolfe's GPS Accessories www.gpscables.com/catalog.html
Kyosho Tracker 2 R/C Kit Servos (2), Batteries	$175	Tower Hobbies www.towerhobbies.com
Vector 2XG Compass	$100	Precision Navigation, Inc. www.precisionnavigation.com
Asst. Electronics/Supplies	$75	Jameco, Radio Shack
Electronic Speed Control	$70	Hobby Store Ocala, FL

Acknowlegements/References

Eric Anderson - provided help with motor driver hardware design, and let me borrow his Vector compass software, which I rewrote in IC.

Dan Williams - fellow student who worked with me on the NMEA 0183 communication interface used in my GPS and his depth finder.

Scott Jantz - Helped with all of the basic questions and helped out anytime I was stuck.

The NMEA FAQ - http://vancouver-webpages.com/pub/peter/nmeafaw.txt
An excellent resource on the NMEA communication interface

Appendix A - Program Code

```
/*********************************************************
**                                      **
** Title:        Object Avoidance.c              **
** Programmer:          Jeff Brigman                    **
**                                      **
** Servo Turret and Sonar Test Program             **
** also motor, bumper, and sonar analog reading        **
*********************************************************/



/******** Defines ***************************/
#define SONAR          analog(0)
#define BUMPER              analog(1)
/***** End of Defines ***********************/


float MAX_SPEED    = 100.0 ;
float ZERO_SPEED   = 0.0   ;
int LEFT = 2200;
int RIGHT = 3800;
int STRAIGHT = 3000;
```

```
int heading;

int distance;

int rand;

int count;

int offset = 45;

int object = 200;



/*****************Servo Routines**********************/

void MOTOR_OFF  (void) { servo2( 3000   ); }

void MOTOR_FULL (void) { servo2( 3500 ); }

void MOTOR_HALF (void) { servo2( 3200 ); }

void MOTOR_REV  (void) { servo2( 2500  ); }


void STEER_LEFT (void) { servo1( 2200 ); }

void STEER_RIGHT (void) { servo1( 3800 ); }

void STEER_STRAIGHT (void) { servo1( 3000 ); }

/*****************End Servo Routines********************/



/**********************INIT_SPI ROUTINE***************/
```

```
void init_SPI()
{
        poke(0x1028,0x4C);     /*set SPCR register */
        poke (0x7000, 0x10);   /*p/c high in reset mode*/
        sleep(1.25);           /* delay */

        poke (0x7000, 0x14);   /*coming out of reset*/
        sleep(1.5);
return;
}
/**********************END SPI************************/



/*****************GET_SPI ROUTINE************************/
int get_SPI()
{
int test=0;
int value1=0;
int value2=0;

poke(0x7000, 0x04);     /* Take P/C low */
```

```
sleep(.02);            /* Hold at least 10ms */

poke(0x7000, 0x14);    /* P/C goes back high */


while (test == 0)

  {

  test = peek(0x1029);

  test = test & 0x80;

  }

value1= peek(0x102A);


test=0;


while (test == 0)

  {

  test = peek(0x1029);

  test = test & 0x80;

  }

value2= peek(0x102A);

value1 = value1 & 0x01;


if (value1 > 0 && value2 < 104)
```

```
   heading = value2+256;
else
   heading= value2;
heading = heading - offset;  /* correct for compass offset */
if (heading < 0) heading=heading+360;
return(heading);
}
/**********************END ROUTINE**********************/



/********************Avoid() Routine******************/
void avoid()
{
        if (BUMPER<200) collision();  /* contact made */
        distance=SONAR;
        if (distance<object)
                {
                rand = peek(0x100F);
                rand = rand & 1;
                if (rand==1) {
```

```
                while(SONAR<object)
        {
    STEER_LEFT();
    sleep(.1);
    if (BUMPER<200) collision();
    }
        STEER_STRAIGHT();
        }
        else  {
                while(SONAR<object) {
    STEER_RIGHT();
    sleep(.1);
    if (BUMPER<200) collision();
    }
        STEER_STRAIGHT();
        }
    }
    else  {
    STEER_STRAIGHT();
      if (BUMPER<200) collision();  /* contact made */
else MOTOR_HALF();
```

```
    }

}

/*******************collision()*****************************/

void collision()
{
MOTOR_REV();    /* rev for braking, backing up */
if (rand==1) STEER_RIGHT();   /* steer away from obstacle */
else STEER_LEFT();
sleep(.5);
MOTOR_OFF();    /* allow time to stop from reverse */
sleep(.5);
while(BUMPER<200) MOTOR_OFF();    /* stay stopped if bumper is stuck */
return;
}



/*******************Track North() ************************/
void track_north()
```

```
{
int bearing=0;
while(1){
        heading=get_SPI();

        if (heading < 180)
                while(heading > 25) {heading=get_SPI(); STEER_LEFT();}

        else
                while (heading < 335) {heading=get_SPI(); STEER_RIGHT();}

        STEER_STRAIGHT();
        }
}

/**********************End of Track North******************/

/**********************Main************************/
void main()
{
```

```
servo_on();
init_SPI();
while (BUMPER>200) ;   /* wait for bumper press */
while (BUMPER<100) ;                /* and release */
MOTOR_HALF();
while(1) {
        track_north();
        count = 500;
        while(count > 0) avoid();
  }
}
```

/**********End of Main**********/

/**********End of file**********/