

# Programmable Self-Navigating Lego Car

ECE430 Final Project:  
Dr. Song  
November 17, 2003

Jeff Keacher  
Joey Richey

## TABLE OF CONTENTS

<b>1.INTRODUCTION.....</b>	<b>1</b>
<b>2.OBJECTIVES AND SPECIFICATIONS.....</b>	<b>1</b>
<b>3.STRATEGY AND IMPLEMENTATION.....</b>	<b>1</b>
<b>4.TESTING PROCEDURE AND RESULTS.....</b>	<b>3</b>
<b>5.BILL OF MATERIALS.....</b>	<b>4</b>
<b>6.USER MANUAL.....</b>	<b>4</b>
<b>7.CONCLUSIONS AND RECOMMENDATIONS.....</b>	<b>6</b>
<b>8.REFERENCES.....</b>	<b>7</b>

# 1. Introduction

---

Our final project for ECE430: Microcomputers was to design and construct a small robot capable of following a human-issued directive to a specified location. We chose this project because we would be able to incorporate motors and robotics in a setting slightly more complex than a black-line-following drone.

The robot was designed to allow the destination to be easily input from a keypad and displayed on an LCD. The device, after gathering input from both the user and the GPS, would then navigate to the desired destination. Once at the destination, the robot would stop.

## 2. Objectives and Specifications

---

The purpose of our project was to make a robot capable of carrying out navigation to a destination using GPS data. A corollary to this is that our project has a high "cool" factor. Due to complexity issues, we chose not to implement obstacle-avoidance routines.

The robot was required to fulfill several requirements, including:

- The robot shall take input from the user via a keypad.
- The robot shall allow the destination to be specified in terms of distance from the current location, in any direction.
- The robot input distance unit shall be feet.
- The robot shall navigate to the final destination using GPS data.
- The robot shall correct its course on the way to its destination, therefore the robot shall be capable of turning.
- The robot does not need to handle obstacles in its path.
- The robot shall halt once it arrives at its destination
- The robot shall constantly display its current and destination locations while in navigation mode.

## 3. Strategy and Implementation

---

We chose to implement our project using a PIC16F877A. We chose this chip because of our previous work with it as well as its easy procurement. For the project, the PIC remained mounted on an X-1 development board. We used this board because it included three components that we deemed essential to our project: a keypad, an LCD display, and a RS-232 serial port.

To navigate to a destination, the robot must be told where the destination is. We programmed the PIC such that this input was gathered using the X-1 keypad and displayed on the LCD. The User Manual elsewhere in this document describes the user interface. Initially, we intended for the user to input the destination in terms of absolute latitude and longitude coordinates. However, this method had two drawbacks: most people are unaware of the latitude and longitude coordinates of anywhere, and the GPS

tends to "drift" over a several hundred foot range, leading to poor accuracy. Instead, by inputting a relative distance in feet from the current location, user input is simplified and accuracy is greatly improved.

A Garmin GPS receiver provides location awareness. We chose the Garmin eTrex GPS not only because we already owned one, but also because it incorporates an RS-232 serial port through which location data can be transmitted. The serial data is sent at 9600 bps in 8/N/1 format. Physically connecting the GPS to the X-1 required a special cable, as the GPS has a proprietary plug instead of a standard DB-9 connector. As both the GPS and the X-1 are set up as peripheral devices, we used a null-modem adapter to align the transmit/receive lines in the serial cable. We capture the data in the PIC using interrupts so that serial data is never lost.

The location data can be sent in a variety of formats. We settled on a standard Garmin format known as "Simple Text Output." In Simple Text mode, data is sent from the GPS once per second in a standard, ASCII text sentence. The sentence contains, among other things, the current location and current velocity. By parsing the sentence for these two values, we determine the distance to the target location and the current orientation of the robot. The latitude and longitude data from the GPS is significant to one thousandth of a minute, which, in Indiana, translates roughly to a six-foot accuracy.

One of our requirements was that the robot be able to propel itself to a destination. We chose to fulfill this requirement by building a wheeled chassis powered by electric motors. For ease of construction, we built the chassis out of Legos and powered the wheels using two Mindstorms motors. We included two motors, one for each of the two driven wheels, with each driven wheel on one side of the chassis. These motors are switched on and off by the PIC using IRF540 Power MOSFETs. These transistors were chosen for their ease of implementation (no biasing required) as well as their robust nature (they are rated to nearly 30 amps).

The robot implements a method to correct its course towards the destination. After the user has entered the desired location, the PIC converts the destination to a latitude and longitude coordinate pair. For simplicity, we chose to express that coordinate in terms of minutes, ignoring degrees. As long as the location is not near 0 or 60 minutes (true in Terre Haute), this simplification should have no effect on navigation. After the conversion from feet to coordinates, the PIC opens the serial port and begins parsing the GPS data. All the while the motors remain off. Once good data from the GPS is received, the PIC turns the motors on and begins moving. The PIC continually checks the position information for signs of change. After every few data points, the PIC evaluates its current course of direction. The velocity data from the GPS, in conjunction with an arctangent function, reveals the current direction of motion of the robot. Concurrently, the PIC calculates the direction of travel necessary to propel the robot to the destination. After both the current and required directions are known, the robot subtracts the two angles to calculate the amount of rotation necessary to point itself towards the target. A turning function takes this rotation angle and switches off one motor for a time appropriate to spin the robot towards the correct direction. After the turn, the robot resumes forward direction and repeats the cycle.

There are several conditions under which the device will halt. As mentioned above, the device will not initially move until it knows the destination and is receiving good data from the GPS. If the GPS should stop sending good data, the robot will halt until the data stream resumes. This precaution prevents the robot from following erroneous data or driving "blind." The robot will also stop once it arrives at its destination. After arriving at the destination, the robot will not move again until a new destination is entered.

We encountered several setbacks during development but triumphed over all. In the early stages of the project, we found that enabling code optimization broke our LCD driver. Transferring the code to a new project solved that problem. Calculation of the direction of travel required an arctangent function. One was available in the ANSI math library, but inclusion of that library bloated the size of the program such that it exceeded the bounds of the PIC. Instead, we wrote a custom arctangent function that compiled to roughly 10% of the size of the library version. Perhaps the most frustrating problem was that of stack overflow. After spending a good deal of time experiencing inconsistent errors, we traced the problem to the PIC stack overflowing. This occurred despite the fact that the compiler should have inlined the culprit function.

## **4. Testing Procedure and Results**

---

We began our testing and debugging by confirming that the robot successfully performed in a static position. We hooked the circuit to an external power supply, inserted LEDs in place of the motors, and moved the GPS receiver to a window (that way the GPS would provide solid data). With this setup, we were able to confirm that the motors would turn on and off at the correct time and that the robot could successfully receive and process the GPS data.

In testing the GPS feed, we compared the position displayed on the X-1 LCD to that displayed on the GPS's internal LCD. The positions matched perfectly. We then confirmed that the destination position was calculated and displayed in accordance with the values entered at the onset of testing. These too matched perfectly.

Satisfied that the robot worked in a static condition, we replaced the LEDs with the Mindstorms motors and mounted the board, a battery and the motors on the Lego chassis. We confirmed that the motors would propel the robot in a straight line.

In the same configuration, we characterized the approximate turning rate by disconnecting one of the motors. That data was used to adjust the timing for the turning subroutine as well as the frequency of turns.

Finally, we connected and mounted the GPS receiver, turned on all of the circuits, and went outside to test the entire device. Unfortunately, the GPS was unable to receive any satellite signals. Confused, we turned the robot off, leaving the GPS on. Almost immediately, the GPS found the satellites and its position. We turned the robot back on, and the GPS quickly lost the satellites.

It appears that the X-1 board emits noise that interferes with the GPS receiver. We attempted to rectify the situation by moving the GPS around on the chassis, shielding the board with aluminum foil, and altering the PIC clock speed. Suspecting that the noise might be going to the GPS via the serial cable, we disconnected the cable. Unfortunately, the noise remained. A slower clock speed combined with extensive foil shielding seemed to help a little bit, but not enough.

We guessed that distancing the GPS from the rest of the robot would decrease the interference. When the GPS was held more than four feet from the board, it captured the satellite signals and calculated position information.

In this configuration, with the GPS held away from the robot by a human, the robot performed as expected. It accepted input, navigated to the destination, and halted upon arrival. Had we access to better shielding, we are confident that the robot would have worked without a human following it around holding the GPS.

## 5. Bill of Materials

---

The total cost of this project remained quite low. Our vast collection of electronic devices significantly curtailed our required spending. As shown in Table 1, our total outlay was quite reasonable.

Item	Quantity	Price (each)	Cost to us (total)
PIC16F877A	1	\$10	0
X-1 board	1	100	0
IRF540 power MOSFET	2	1	2
Misc. wire	3ft	1	0
Lego Mindstorms motor	2	10	0
Lego chassis	1	20	0
GPS receiver	1	100	0
GPS interface cable	1	20	20
9.6v rechargeable battery	1	20	0
Coaxial barrel-type power connector	1	1	1
		Total	\$23

Table 1: Bill of Materials

## 6. User Manual

---

To make the car navigate, the user enters the desired number of feet to go in both a north/south and east/west direction. After that data is entered, the car will begin to move.

It will continue to drive, occasionally stopping to turn and correct its course, until it reaches its commanded destination. In addition to halting, the car will display a message upon arrival at its destination.

1	2	3	N/E
4	5	6	S/W
7	8	9	
0	B	E	

**Figure 1: Keypad Layout**

When the car is powered on, the LCD prompts the user to enter the destination. Figure 1 shows the layout of the keys on the car for data input. Table 2 describes the key usage.

<b>Key</b>	<b>Function</b>
0-9	Numeric Data Entry
B	Backspace
E	Enter
N/E	Direction; North during Latitude entry, East during Longitude entry
S/W	Direction. South during Latitude entry, West during Longitude entry

**Table 2: Keypad Functions**

The first information that must be entered is the north/south delta. Use the numeric keypad to enter the number of feet you wish to travel (it must be less than 255). If you make a mistake you can press the backspace key to correct it. After entering the magnitude of the direction, press the North or South button to indicate if the direction is North or South. After all the information is entered correctly press Enter.

You will then be prompted for east/west delta. The procedure to enter this data is the same as for the north/south data, except the directions will be East and West.

After the complete destination is entered, the car will attempt to read a signal from the GPS unit. The screen will begin showing the current and destination positions once the position is known. If the screen remains blank at this point, the GPS is not receiving good position data. Ensure that there is a clear path to the sky if this occurs.

If the data is good, the LCD will display pertinent information, as shown in Table 3 and Table 4. The location information is displayed in terms of minutes of degrees, accurate to one thousandth of a minute. On screen, a decimal point is implied in to the right of the second most significant digit.

The speed number format is the 2's complement number of the speed, where a negative number indicates the direction is South or West and a positive number means North or East.

AAAAA	BBBBB	EEEE
CCCCC	DDDDD	EEEE

**Table 3: LCD Status Screen Organization**

<b>Number Group</b>	<b>Meaning</b>
AAAAA	Current Latitude coordinate, in thousandths of minutes
BBBBB	Current Longitude coordinate, in thousandths of minutes
CCCCC	Destination Latitude coordinate, in thousandths of minutes
DDDDD	Destination Longitude coordinate, in thousandths of minutes
EEEE	Current speed in north/south direction
FFFF	Current speed in east/west direction

**Table 4: LCD Screen Symbol Key**

The LCD will continually update as the car moves. If for some reason the car loses the GPS signal, it will stop.

Every ten seconds, the car will stop and turn to realign itself with the destination.

After the car has traveled to its desired location, it will stop and display a message on the LCD indicating that it has finished. The car is limited by the accuracy of the GPS, about six feet. The implication of this accuracy limitation is that the car may halt up to six feet from the true destination.

Up to and including the time of arrival at the destination, the robot car's reset button may be pushed to cancel the current navigation and enter a new destination.

## **7. Conclusions and Recommendations**

If we were able to solve the problem of electromagnetic interference from the X-1 board, we feel that we could have made a fully functioning GPS-navigated car. All of the individual parts were implemented and tested successfully separately. When we did simulated driving (by holding the GPS unit away from the car), the car shut off its motors at the correct location.

Although we were unable to make the car fully functional as a self-contained unit, we still implemented all the parts of the program and utilized many of the ideas we covered in class: RS232 communication was used to gather data from the GPS unit; the keypad was used for gathering information from the user; the LCD displayed information useful to the user and for debugging purposes. We were also able to interface with external motors and construct a physical chassis to carry our electronics.

Our project would have benefited if we could had used a board that we could modify. Since we used the X-1 board, which belongs to Rose-Hulman, we were unable to solder anything to it. This meant that many of wires were connected to the board in an insecure manner. Had we been able to solder wires to the X-1, the car would have been more durable.

Were we to do the project over, we would build a complete prototype earlier in the development phase. We built our project in modular components, testing each one as we went along. It was not until the near the end of the project that we mounted all of the parts onto the car as a single unit. If we had assembled the components into a complete system earlier in the project, we would have noticed the electromagnetic interference earlier, which might have allowed us to fix the problem.

## **8. References**

---

Reference Material for ECE430 Microcomputers. Compiled by Jianjian Song. Sept 2003.

<http://www.convict.lu/Jeunes/Math/arctan.htm> (Polynomial approximation of arctangent)