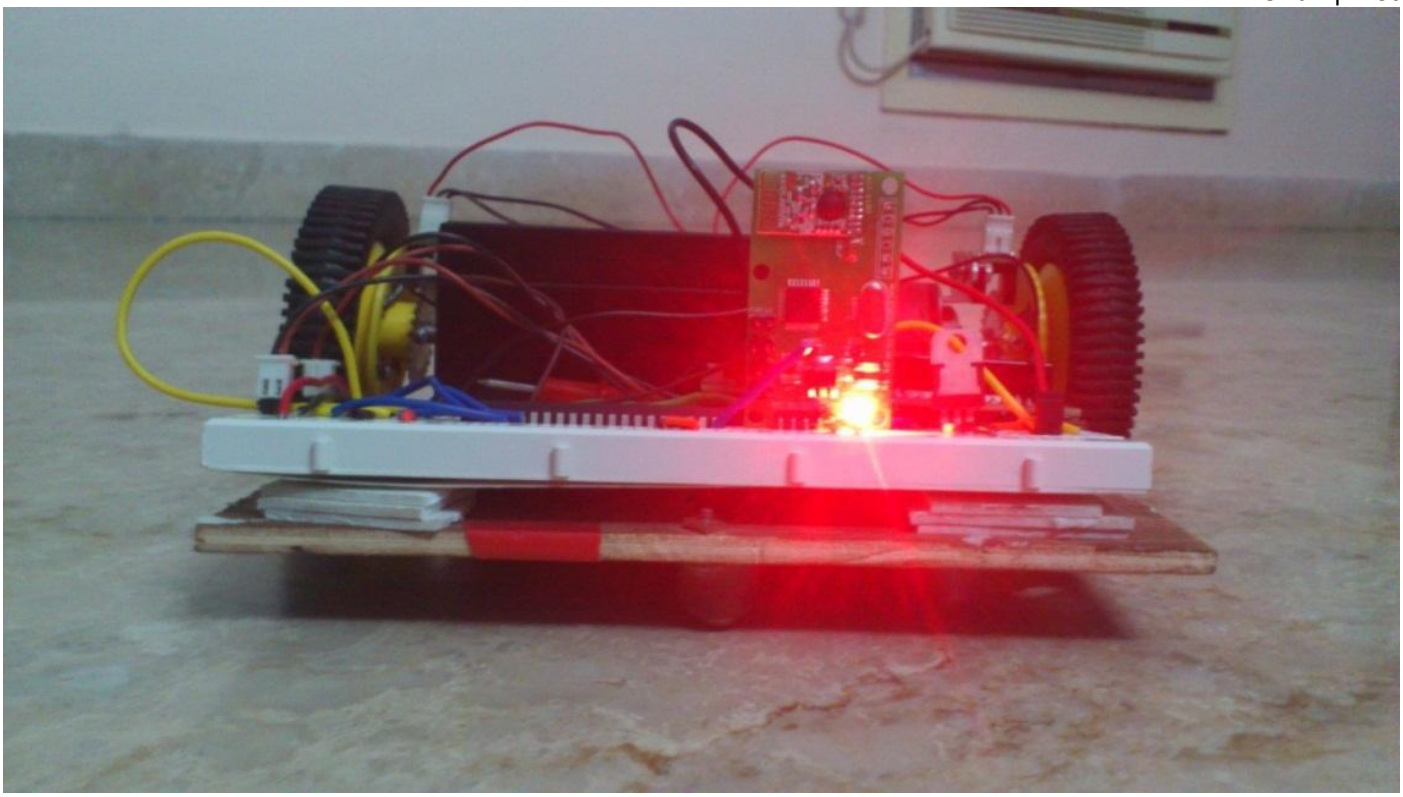
Indian Institute of Technology Madras

Centre For Innovation

SNav

Documentation



INDEX

1. Abstract and Motivation

This project focuses on developing a *Smart Navigation* system for bots as an direct alternative to presently used *Line Followers*. Having stored into memory specific details of a path previously tread, the unit will then be able to retrace the route. A number of discrete paths can also be efficiently handled. Having created a virtual map of the paths it has learnt, the unit is capable of finding the shortest path to its destination. Furthermore, S-Nav has been primarily constructed on a differential drive system.

2. Team Members

NAME	ROLL NO	EMAIL
Karthik M	ME11B153	karthik1708@gmail.com
S R Manikandasriram	EE11B127	srmanikandasriram@gmail.com

3. Concept

What is Smart Navigation?

S-Nav is a system which enables robots to remember a traversed path and guide themselves through it any number of times independently.

Why do we need S-Nav?

A large majority of contemporary bots can be classified into two types based on their navigational abilities: Externally dependent and continually controlled.

S-Nav is neither. It is a hybrid.

The purpose of this document is to help you familiarize yourself with S-Nav, its requirements, its working and its abilities. Starting with the objectives, scope and limitations, and then moving on to the concepts involved and the core algorithm, this document will bring to light the depth and effectiveness of S-Nav.

The references and resources section has a list of relevant websites, electronics shops and commonly used components.

4. Objectives

S-Nav has a single primary objective: To take wayfinding to the next level by encapsulating the navigation system in a single unit and in the process removing all external directional aids what-so-ever.

Furthermore, having recognised several distinct paths which are *taught* to it, the unit aims at successfully calculating, from a self-generated map, the most efficient manner of traversing from one place to another.

5. Mechanical Structure

• Ground Work

In order to setup a strong foundation for S-Nav, ideas and concepts regarding usage and implementation of encoders were first researched. Also, at that time, incorporating an optical mouse was also contemplated. A sound basis of the core algorithm and working was constructed and refined over a few days. Extensive research was also done on computation of distance and direction. A strong idea of the components was generated and discussed. Overall, the groundwork done was a sufficient kick-start to S-Nav.

• Entities Involved

The controller

It is a handheld object consisting of an LCD, a keypad and powered by a 9V battery. It is used to direct or command the bot which it accomplishes using an RF link.

The bot

It is the on-ground movable unit constructed as a simple differential drive. Once made to trace a path, it remembers the details of the path by storing it in its memory and can successfully retrace a queried path.

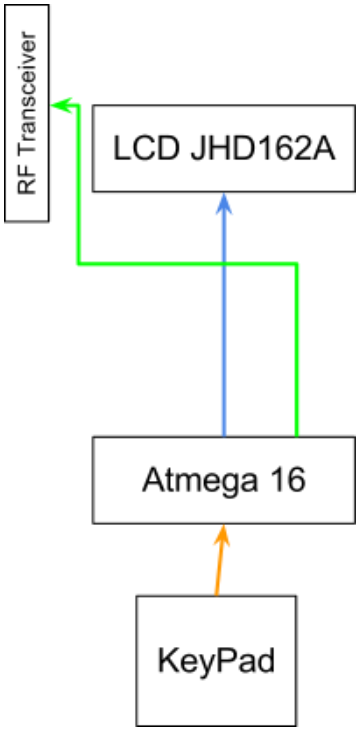
6. Electronics

Topics Involved

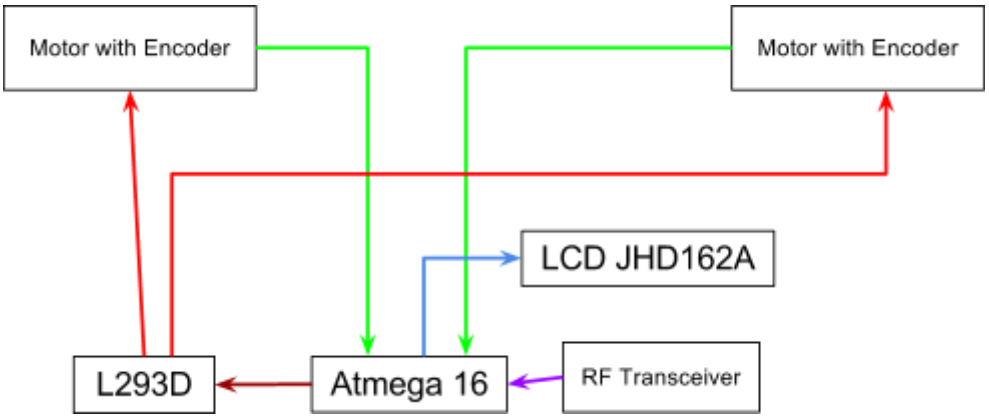
- Making of a USBTiny Programmer
- Motor Driver L293D usage
- PWM incorporation
- USART interfacing
- LCD interfacing
- Keypad interfacing
- EEPROM usage
- Interrupts and timers
- Encoders
- Rigorous memory management

Schematic

The Controller



The Bot



7. Coding

Methodology

The very first point in spacetime where the bot is powered on is named “Origin”. “Origin” is thus our first checkpoint. A path is defined by two checkpoints - Source and Destination. When you steer the bot through a path, the two encoders continuously raise interrupts and these details are present in the buffer memory. On reaching the destination, you will have to create the Destination Checkpoint thereby teaching the bot a path between Source and Destination. You could also select the destination as a previously stored Checkpoint, thereby teaching the bot multiple paths to reach a destination. The Smartness of our navigation system is realised fully when you see the bot always picking the shortest route to its destination. The path details from the buffer memory are now transferred into the EEPROM and stored in a specific format, which will be explained in the MPR Architecture section. In this way, you can teach the bot any complicated graph made up of any number of loops, branches, and checkpoints.

The smart bot is always ready to serve you. You can now command the bot to go to any place that you have taught it before at the press of a button. The bot immediately switches to “auto-pilot” mode and powers its motors to reach the destination through the shortest path possible.

Algorithm

MPR Architecture:

Here, we will explain the algorithm involved in converting the pure plain interrupts that the encoders raise in the microcontroller into more meaningful path details. Since our project is memory intensive, we have developed an efficient way to handle the limited 512 bytes of EEPROM memory that an Atmega16 provides.

The EEPROM is virtually partitioned into 3 tables - The Master Database(MDB), The Path-details Database(PDB) and our own Router Database(RDB). As these three tables form the basis of our memory management, we would like to refer to this as the MPR Architecture.

Block-and-Reminder system:

In order to utilise the meagre 512 bytes of Atmega16 efficiently, we save numbers larger than 255 using a chunk of two bytes which are the quotient(or block) and reminder obtained on dividing the number by 255. This facilitates reading and writing to EEPROM.

The Master Database (MDB) :

Address (0-indexed)	Value	Description
0	posm	The address of the end of MDB
1	posp/255	The address of the end of PDB as block and remainder
2	posp%255	
	Path header starts here	

3	Start	The index of the starting checkpoint of the path
4	End	The index of the ending checkpoint of the path
5	path_dist/255	The total distance of the path stored as block and remainder
6	path_dist%255	
7	Start Angle	The starting angle of the path
8	End Angle	The ending angle of the path
	Path header ends here	

The Path-details Database (PDB) :

The Path details are stored in a specific format. Forward is considered as the default direction of travel. The Left turn and Right turn are thus marked using “Markers”, which is nothing more than a predetermined character viz Left turn is marked by 250 and Right turn is marked by 251. The PDB extends from address 100 to address 399 in the EEPROM.

Consider the following to be the path details of the first path the bot is taught.

1. Move forward by 130 encoder counts.
2. Turn right by 50 encoder counts.
3. Move forward by 275 encoder counts.
4. Turn left by 100 encoder counts.
5. Move forward by 500 encoder counts.
6. Stop and Save.

The corresponding entries in the PDB would be:

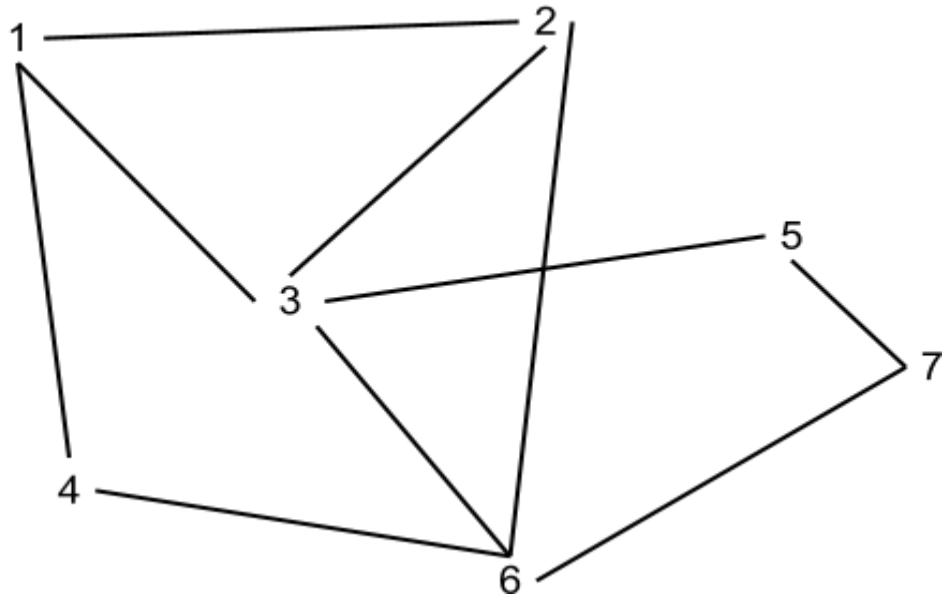
Address	**Value**
100	000
101	130
102	251
103	0

104	50
105	1
106	20
107	250
108	0
109	100
110	1
111	245

The Router Database (RDB) :

This DB is the first part of the Shortest Path Algorithm. This DB contains the map data in a specific format. Every Checkpoint can be connected to 5 other checkpoints *(restriction due to memory constraints and ease of coding)*. The RDB starts from address 401. The neighbors of Origin are stored from 401 to 405 (both inclusive), the neighbors of CheckpointA are stored from 406 to 410 (both inclusive) and so on.

Consider the following graph has been taught to the bot,



The RDB corresponding to the above graph will be as follows:

Address	Value
401,402,403,404,405	2,3,4,0,0
406,407,408,409,410	1,3,6,0,0
411,412,413,414,415	1,2,5,6,0
416,417,418,419,420	1,6,0,0,0
421,422,423,424,425	3,7,0,0,0
426,427,428,429,430	2,3,4,7,0
431,432,433,434,435	5,6,0,0,0

Shortest Path Algorithm

Note: As we could not find any algorithm on the Internet which met our specific needs, we have developed our own algorithm for the same.

The first step towards finding the shortest path to the destination is to find all the routes that lead to the destination from the bot's current position.

Note: A path is between two checkpoints. A route, on the otherhand , is the sequence of paths that lead to the destination from the source.

In order to find all the routes to the destination, we need to travel through the graph with the RDB to guide us. Starting from the Current Position, we need to choose one among the neighboring checkpoints and continue to do so until the destination is reached.

There needs to be three checks performed before a checkpoint is chosen. They are:

- The checkpoint must not be already present in the current route.
- The checkpoint must not be a deadend.
- The checkpoint should not have been chosen the last time the current checkpoint was visited through the current route.

On reaching the destination, add the route to the array of routes found and continue to search for the next checkpoint satisfying the above three checks from the penultimate checkpoint. When no more checkpoints can be chosen, take a step back on your current route and continue the search. Continue this iteration until you exhaust all the neighboring checkpoints of the starting checkpoint.

Once we have found all the routes, we can retrieve the path lengths of individual paths which constitute the route from the MDB and find the route distance for all the paths.

8. Budget Details:

Sr No.	Item	Acquired from: Link(If item is online), Chennai shop name	Amount (Rs.)
1	IR Transceiver	www.robokits.co.in	1700
2	Wheel Encoder Set	www.nex-robotics.com	500
3	Elec and Mech Items for the Bot	Modi Electronics, Ajanta Electronics, Ritchie Street	2000

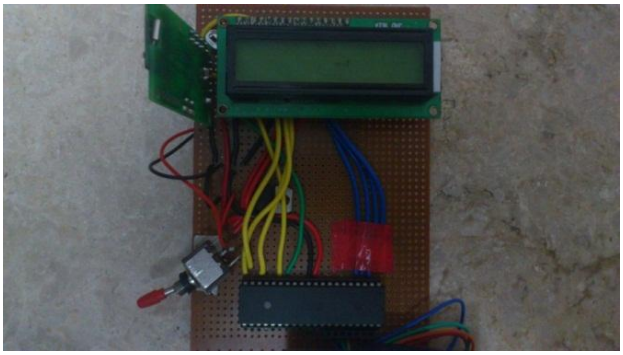
9. Problems Faced

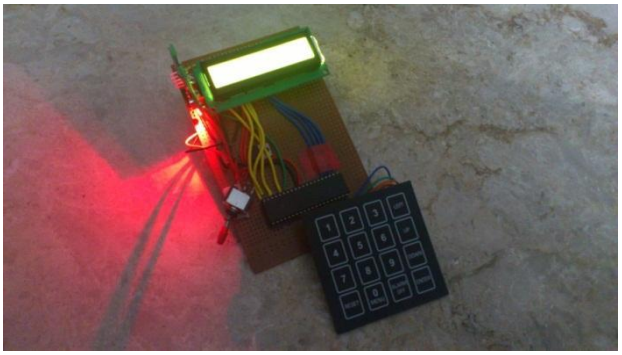
- Most batteries unable to withstand current drawn from motors for extended periods of time.
- Some breadboards were faulty.
- Bur connectors didn't match perfectly.
- The wiring was messy initially which made it difficult to place and remove micro during coding.
- Encoder sensors were displaced and misaligned due to the motion of the bot.
- Encoder wheels and the main wheels were not exactly parallel always.
- Mounting of breadboard was unstable at times.
- RMC connectors to motors had not been soldered properly.
- The first algorithm for shortest path was extensively applied but unsuccessful.

- Keypad on controller was not mounted properly which made it inconvenient to handle.
- Keypad wires were very thin and therefore soldering had to be perfect.
- Soldering of LCD to pcb was tedious and resulted in short-circuits.
- A lot of time initially was spent on LCD and keypad integration.
- Used non-rechargeable 9V batteries for controller.

- Did not insist on reimbursement until very late.
- Had difficulty in accessing material quickly as work had mainly not been done in cfi.
- Simple mistakes with respect to wiring occurred and ready help and expertise was unavailable.
- Multimeter probes were not handled with care.
- LCD connections were often confusing and modifying lcd.h was necessary.
- The motor driver proved to be very sensitive to wrong connections and was hence damaged.
- Encoder screws were tiny and easily misplaced.

10. Pictures and Video Links





Here are the YouTube video links:

http://www.youtube.com/watch?v=8bbGucClAPc

http://www.youtube.com/watch?v=5lzlkC7Ur2k

11. References

Links:

For ideas regarding methodology and algorithm

- www.wikipedia.org

For help with hardware and compatibility

- www.extremeelectronics.co.in
- www.robokits.co.in
- www.nex-robotics.com

Other resources:

Tools:

- AVR Studio 4
- Programmer's Noteped 2
- Makefile
- USBTiny Programmer

Components Used:

- 2.4Ghz RF Transceiver module
- Wheel Encoder Set
- JHD162 LCD module
- 4 X 4 Matrix Keypad
- Atmega 16
- L293D
- 7805

- 12V Battery
- 200 RPM 12V DC motors
- Dot Matrix Printed Circuit Board
- Breadboard

Shops:

- Modi Electronics, Ritchie Street
- Ajanta Electronics, Ritchie Street

12. Data Sheets

For help with programming and using atmega16

- www.atmel.com
- www.avrfreaks.net
- www.winavr.sourceforge.net
- www.engbedded.com/fusecalc
- www.robokits.co.in
- www.nex-robotics.com

13. Acknowledgements

Firstly, we would like to thank Cfi for providing us an excellent atmosphere to work in and easily accessible material at all times. Special thanks goes to the entire Ibot team who always helped us with issues that were tricky and difficult to solve. We are also deeply grateful to our project mentors Siddharth Dialani and Varun Nalam for their continued support and encouragement.