All-Terrain Network Service Robot Based on Tekkostu Framework

Derrick Jones, Ju Wang, John Lewis
Math and Computer Science Department
Virginia State University
1 Hayden Dr
Petersburg, VA 23806

Abstract

We report the design and implement of a land robot whose primary task is to "patch" a Wireless Sensor Network. the Tekkotsu/Create system is modified to support GPS-guided navigation and radio-sensing based navigation . The project is a moderate success with both GPS and radio-sensing navigation algorithms achieve similar navigation performance.

1. Introduction

The advances of robotic technology in the past decades have results in significant increase of its usage throughout industry and manufacturing. In additional to specialized industry robot, "general purpose" robots that are capable of accomplish more complicated tasks are attracting more attention. One particular example is land robot system such as unmanned ground vehicle. The usefulness of land robot in supporting military applications is obvious, as evident by the fast adoption of anti-IED robots in military forces. In civilian world, such robot also proves to be valuable to perform surveillance and other service.

In this paper, we report the design and implement of a land robot whose primary task is to "patch" a Wireless Sensor Network. The project originated from the need to provide a quick fix to the High Tunnel greenhouse Wireless Sensor Network at VSU's Randolph Farm. The network was deployed in 2009 to collect critical growth data in the greenhouse environment. From time to time, the network would not be accessible due to a failed relay node or poor signal quality. The robot discussed here would be used as a mobile service node that could fill the hole in the network.

The main task is formulated as a guided-navigation problem: given the rough location of the network outage, navigate the land robot to the problem area to re-connect the network. Due to the uncertainty of the problem location, the robot must explore the area to find a perfect relaying point.

The robot chassis is built upon a 6-wheel platform. Each wheel is driven by a gear motor and independently controlled. The low level driver is implemented at a Cyclone II FPGA to provide a rich set of driving patterns allowing the robot to maneuver through rugged terrain. High level navigation, as well as communication and task dispatch, is implemented in the Tekkotsu framework.

The rest of the paper is organized as following. Section 2 presents the background and related works. Section 3 presents the design of the low level driver-chain. Section 4 discusses the navigation algorithm and upper layer software architecture. Section 5 presents our preliminary experiment results.

2. Background and Design Overview

The Greenhouse Wireless Sensor Network consists of four EcoMote sensing nodes and one base station. The sensor nodes measure the ambient temperature, moisture level, and soil water content level at several locations inside the greenhouse. Each sensor node is equipped with a ZigBee wireless module which allows nodes to exchange data. The data collected is relayed to the EKO Pro Series gateway, located in an office about half mile away from the greenhouse. Figure 1 shows the data accessed through the embedded web-server at the gateway.

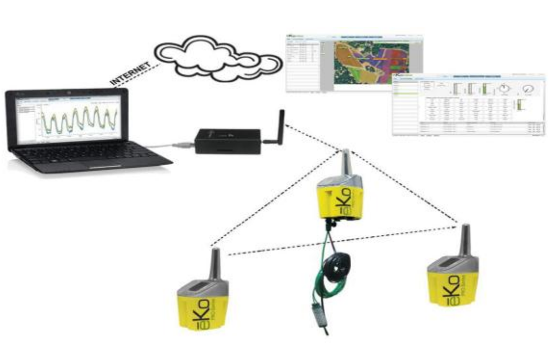

Figure 1: wireless sensor network

The fundamental architecture of the PatchBot is of the Tekkostu/Create design. The brain of the robot is based on Tekkotsu, an open source, event-based architecture. The

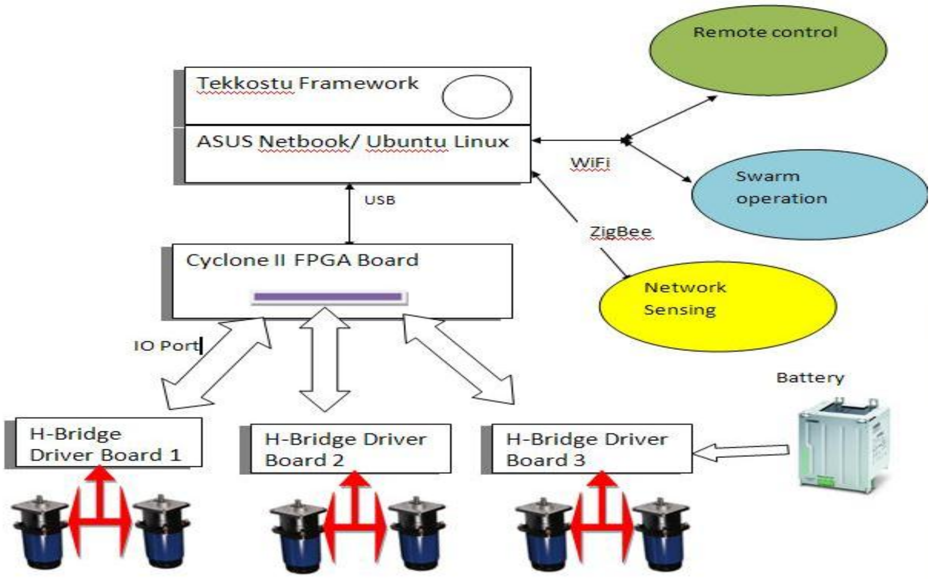

Figure 2: PatchBot architecture

Tekkotsu framework support several hardware platforms with different actuators configuration. The most widely seen hardware platform is ICreate by iRobot, which features two wheels, a rotational disk, and various bumper sensors. In a pilot study, the Tekkotsu/Create system is modified to support GPS-guided navigation. The project is a moderate success with a working GPS navigation algorithm. However it also showed that the iRobot platform is severely limited in an unpaved environment. Such is one of the motivations to port the Tekkotsu to a more powerful, all-terrain platform. The driving platform is more powerful than the CREATE robot with six gear motors and a high chassis designed for unpaved terrain. Figure 2 shows the overall architecture of the system.

The FPGA device generate control signal that turn on/off motors. VHDL's and Verilog are two standard forms of hardware design language supported by FPGA. Verilog is easier to learn then VHDL because it looks like C programming in syntax and is widely used in the industry. The main advantages that Verilog has over other modeling circuit systems is that is powerful enough to express complex testing procedures without restoring to a different language.

3. Driver-Chain Design

Driver chain is the subsystem that interfaces between high level driving command and the physical actuator (e.g., motors). This layer could be implemented in many different ways and there are many off-the-shelf solutions. Most designs use a micro-controller to provide on/off PWM con-

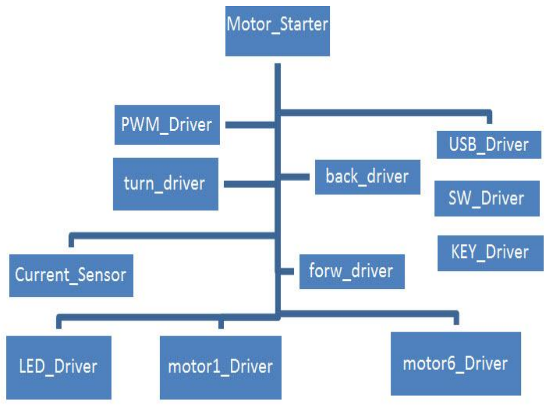

Figure 3: FPGA driver code flow chart

trol signal and power-transistors to energize the motors. Instead of using existing micro-controller based design, we decide to create a new driver chain based on an FPGA device due to several reasons:

- The FPGA device has much more general purpose IO ports than most micro-controller, which allows sufficient room to accommodate new actuators.
- The reprogrammable IO port allows quick modification of the functionality.
- Sophisticated and precisely coordinated motion command can be easily implemented in FPGA. For example, single-step six wheels in a specific sequence.
- Communication between the main controller and the driver chain are much simplified through the USB port.

3.1 Functional modules

The robot that is being used has 6 wheels instead of the standard 4 you might find on an everyday vehicle. This means that turning will require more precise programming going to each of the wheels so as not to over/under turn or get stuck in the process. The following picture details the modules within the verilog code and shows how the different components interact with each other.
After power up, the driver waits for motion commands from outside world. The part that is illustrated as the USB driver will receive commands from the upper level controller (Tekkostu). Other control is provided by monitoring toggle switchers and push buttons on the Cyclone II board. The supported USB commands use a simple format:

Cmd code	Cmd length	Para 1	Para 2…	Para N

For example, the left turn command could be accomplished by turning the right side wheels and left side wheels at the opposite direction. With three parameters this command could be expressed as {LT, 3, 10, 20, 5} where the first parameter of 10 indicate the speed of the positive turning wheel, the second parameter (20) indicate the speed of the negative turning wheel, and the last parameter denote the duration of this command.

The corresponding driver in Tekkotsu has to initiate specific behaviors and calibration of the motion parameters. Further discussion at the Tekkostu level is provided at section 4.

3.2 Motor Speed Controller

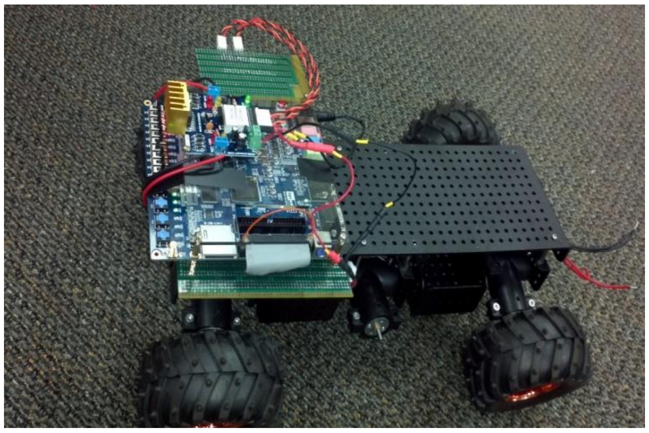

Figure 4: PatchBot: driver chain test with four wheels mounted

A Pulse Width Modulated motor controller with a proportional closed feedback is implemented. The duty cycle of the PWM control how much current is allowed to pass through the motor. However the correlation of the PWM duty and the motor speed is not fixed. There as well must be a speed monitor in a field programmable array.

The proportional controller is used to adjust PWM output so the output speed that is displayed matches the input.
The PWM was created entirely in Verilog, 8 bit switched were used as controller inputs. Current sensor resistors were used to measure the actual current that passes through the motor driver (and motor). The voltage drop on the current sensor is AD converted and input to the FPGA. On the experiment the motor had a 1000 pulse/rotation encoder.

4. Navigation Implementation

We now shift our focus to modification in Tekkostu framework in order to support our mission. The first task is to port the Tekkotsu software to work with the FPGA-based driver-chain. The second task discusses a high-level navigation algorithm that use GPS or network sensing as the navigation guide. The navigation algorithm guides the robot to the location of interest, which in our case is a location near the failed node. Two navigation algorithms will be discussed: the first one assumes the GPS ordinance of the target location is known, and the second algorithm does not assume any prior knowledge of such. The GPS ordinance of the wireless sensor nodes are documented at network planning stage. Hence the GPS based navigation is relatively simple as long as the node ID is provided.

4.1 Determinate the Target

A common task in all navigation algorithms is the identification of the target node. In order to obtain the failure node's ID and its GPS ordinance, a broad-first search must be carried out at the WSN base station. The search will start from the base station node, and check if its direct neighbor is alive or not. If one of the neighbor nodes is found dead, the search will terminate. Otherwise, all neighbors will be inserted to a FIFO queue to continue the search. To determine if a node is alive, one need to query the local data log maintained at the base station, which contain all data reported from the network. In normal operation at our network configuration, each node will report its sensed data every T=30 seconds.

4.2 GPS based navigation

Once the target GPS is obtained, the navigation algorithm could be implemented as a feedback algorithm which constantly compare the current trajectory of the robot and adjust the wheel speed. Let the target coordinate be o=(x,y),

```
WSNtopo g; // global network topology
Node b;        // base station  node
Log   log;     //  accumulated data log
Node FindFailedNode()
{
            Queue q;
            q.enqueue(b);
            while(!q.empty())
            {
             Node x = q.dequeue();
             Bool alive = Log.checkalive(x, T);  // check if
x reported
                        // data in the last T seconds
             If (alive){
                for each node y neighboring with x,
                q.enqueue( y);
             ]else{
                return x; //   find our failed node
             }
            }
```

the previous local coordinate be p=(xp, yp), and the current local coordinate be c=(xc, yc), the navigation algorithm will estimate the errors of its current setting and adjust the wheel speed in next cycle of driving.

Algorithm GPS-Nav:

(a) Estimate the angular offset using equation

$$\cos\alpha = \frac{d^2(p,c) + d^2(p,o) - d^2(c,o)}{2d(p,c)d(p,o)}$$

(b) Calculate the sign

$$\text{sign} = \frac{yc-yp}{xc-xp} - \frac{y-yp}{x-xp}$$

(c) If coos(a) < 0 the trajectory must be reversed,

$$S_l = -S_l, S_r = S_r$$

(d) If sign < 0, the trajectory is offset to the right hand,

$$S_r = S_r + \Delta S$$

(e) If sign >0, the trajectory is offset to the left hand,

$$S_l = S_l + \Delta S$$

(f) Send the driving parameter to the low level driver, and let drive for 5 second.

(g) Check if the distance to the target is within the allowed error, if yes, terminate, else repeat (a-g).

The result of the GPS-based navigation algorithm is given in Figure 4. The error is the locally measured error compared to the provided GPS location. Each sampling point represents the measured error of one iteration of algorithm GPS-Nav. The field measurement indicates that an error margin of 20 meter is normal. The data suggested that GPS navigation is able to could accomplish acceptable navigation accuracy for our application.

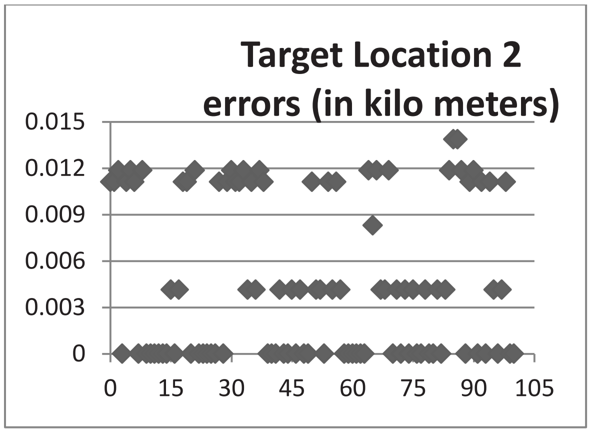

Figure 5: GPS navigation accuracy 5 meters, and avigation algorithm is set to non-stop mode.

4.3 Network sensing based navigation

When the target node GPS coordinate is not available, which is likely the case when large number of nodes are deployed, robot navigation must rely on the network topology and sensing the radio signal strength from node transmissions within the range of the PatchBot.

Network sensing utilizes the received signal strength indicator (RSSI) field in each received packet. RSSI measures the power of the signal at the receiver and based on the known transmit power, the effective propagation loss can be calculated. RSSI is inversely proportional to the distance powerd^n, where d is the distance to the signal source and n>2 is the power factor. If the transmitting wireless sensor node is within the communication range of the robot, the gradient of RSSI between the robot and the signal source (target wireless sensor) is calculated and used as the driving direction. If the signal source can be hear from multiple (neighboring) nodes, the location could be triangulated in a similar manner as localizing mobile phone in cellular network (Ahmed 2005, Bachrach 2005). At high

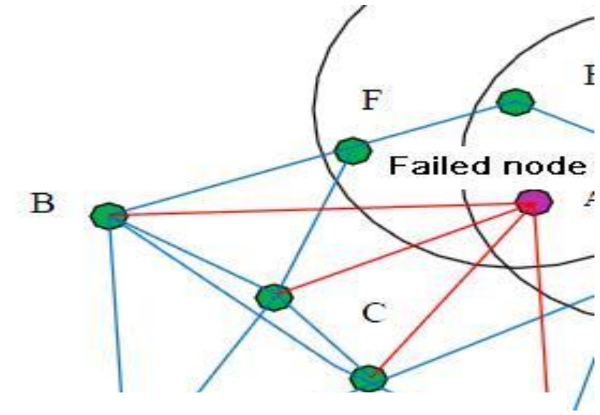

Figure 6: triangulation target location from multiple nodes

level description, this could be done through the following three steps:

(1) Identify the failure node u and the set of nodes W neighboring with u
(2) Find a suitable path for the robot to approach the target hop by hop. This stage will navigate the robot to a close range from the failed node
(3) Estimate the target location by networked sensing and navigate to the location that minimize a cost function

$$\min f = \sum_{j=1\ldots W} (\mathrm{rssi}_{c,j} - \mathrm{rssi}_{o,j})^2$$

Here node $j \in W$ is a node that neighbors the failed node, $\mathrm{rssi}_{c,j}$is the RSSI measurement between the robot and j, and $\mathrm{rssi}_{o,j}$is the RSSI measurement between the failed node and j.

Note that $\mathrm{rssi}_{o,j}$is history data from the base station log while $\mathrm{rssi}_{c,j}$ is the realtime measurement as the robot navigating toward the target. The collaborative nodes enables sensor nodes to accurately estimate their locations by using known beacon locations that are several hops away. We are still conducting field test for this algorithm. However the simulation results based on actual field RSSI data are presented here. The worst case error is 17 meters from the actual location.

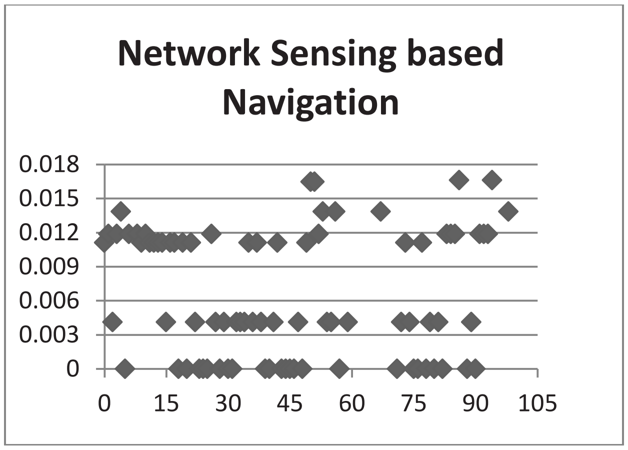

Figure 7: Simulated result for Network Sensing based navigation

5. Conclusion

A field robot to reconnect a broken wireless sensor network is implemented based on Tekkostu framework and a six wheel chassis. The low level driver chain is presented. Two navigation algorithms are implemented, with one algorithm based on GPS coordinate and the other rely on network sensing. Both algorithms demonstrate satisfactory navigation accuracy for our task.

References

T. Oka, M. Inaba and H. Inoue, 1997, Describing a Modular Motion System based on a Real Time Process Network Model, in Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems.

Tekkotsu Wiki: Main Page, http://wiki.tekkotsu.org/index.php/Main_Page.

Cyclone II FPGA Starter Development Board Reference, http://www.altera.com/literature/lit-cyc2.jsp

Are Military Bots the Best Way to Clear Improvised Explosive Devices? http://www.scientificamerican.com/article.cfm?id=robot-ied-clearance.

Frederic G. Snider, R.P.G., GPS: Theory, Practice and Applications, Part I, http://www.PDHonline.org.

A. A. Ahmed, H. Shi, and Y. Shang, 2005, Sharp: A new approach to relative localization in wireless sensor networks," in Proceedings of IEEE ICDCS

J. Bachrach and C. Taylor, 2005, Localization in Sensor Networks, in Handbook of Sensor Networks: Algorithms and Architectures.

Sebastian Thrun, Dieter Fox, Wolfram Burgard, Frank Dellaert, 2001, Robust Monte Carlo localization for mobile robots, Artificial Intelligence: 128, 99–141.

Ndjeng, A.N. 2007, A Multiple Model Localization System for Outdoor Vehicles, in Proceeding of Intelligent Vehicles Symposium.