Debugger for Blinky Blocks

Mihir Rajesh Dattani

Department of Electrical and Computer Engineering

Carnegie Mellon University

Contents

Overview

This document describes the implementation of the debugger for the Blinky Blocks and how to use it.

Project Objective

1) To develop a debugger front end and back end for a system of modular robot blocks.

2) To develop reliable communication (framework) between host and robots as well as among robots to disseminate debug commands and retrieve debug information.

3) To implement debug features on the robots.

Design Overview

Front End: Developed a web based front end capable of generating debug commands to the robots and displaying debug information retrieved from robots

Front End - Robot interface: Developed a multi-threaded server that communicates with the robots on one thread and interacts with the web client on the other thread.

Debug APIs on the robots: Developed and installed APIs for the robots to perform various debugging actions and communicate them to the front end.

Components

Web Front End

File name: jquery.html

Path: oldbb/build/src-bobby/debugger

Description

There are four things that the front end needs to do.

1) Query the blocks to get information of the block configuration
2) Dynamically create place holders for the different blocks in the system
3) Query the blocks for debug messages
4) Update the debug messages of each block in the correct place holder.

Querying the blocks for the block configuration and count is being done by generating http requests to the server by the following lines of code that create html requests to the tiny server which the server interprets. The server then queries the blocks in order to receive the requested information

```
$.getJSON("/a/num_tree",function(data,status){
```

To this the server responds with the following json object

```
{"count":"3"}
```

To query the blocks that constitute the network the query is generated by

```
$.getJSON("/a/num_attendance",function(data2,status){
```

To this the server responds with the following json object

```
{"blocks":[32915,26,9]}
```

Dynamically creating place holders for the debug messages from the blocks is done by the following lines of code

```
$.getJSON("/a/num_attendance",function(data2,status){
    console.log(data2);
    var i;
    for(i = 0 ; i < data2.blocks.length;i++){
    $( "#TEXT" ).append("<div id='block"+data2.blocks[i]+"' class='block'></div>" );
    }
    });
```

This creates div fields in the html file that can be addressed using the block id.

Example:

If the JSON object retrieved in for the /a/num_attendance query is {"blocks":[32915,26,9]} then it will create the following divs in the html

```
<div id="TEXT">
<div id="block0"></div>
<div id="block32915" class="block"> </div>
<div id="block26" class="block"> </div>
<div id="block9" class="block"> </div>
</div>
```

The blocks are queried using the following lines of code

```
$.getJSON("/a/debug_logs?",function(data,status){
```

This retrieves the debug messages from the blocks as JSON objects

They can be empty if no log has been received between successive queries like the JSON object below

{"messages":[{}]}

Or they can be an array of messages generated between successive queries which is terminated by an empty message as in the JSON object below

{"messages":[{"block":"32915","msgid":"9","msg":"Hello"},{"block":"26","msgid":"5","msg":"Hello"},{}]}

The different block place holders are updated with the following lines of code

```
var myVar=setInterval(function(){update_logs()},1000);

function update_logs() {
        $.getJSON("/a/debug_logs?",function(data,status){
                var i = 0;
                if(data.messages.length != 1){

                for(i = 0 ; i < data.messages.length-1;i++){
                console.log(data.messages[i]);
                $( "#block"+data.messages[i].block+"" ).append("block
                "+data.messages[i].block+" msgid "+data.messages[i].msgid+" msg
                "+data.messages[i].msg+"<br>");
                }
                }
                });
}
```

This places the different messages in the correct place holders. The place holders now look like

```
<div id="TEXT">
<div id="block0"></div>
<div id="block32915" class="block">
block 32915 msgid 6 msg SA<br>
block 32915 msgid 7 msg PR<br>
```

*block 32915 msgid 8 msg rs
*

*block 32915 msgid 9 msg Hello
*

*block 32915 msgid 10 msg Hello
*

* *

* *

*block 26 msgid 3 msg PR
*

*block 26 msgid 4 msg rs
*

*block 26 msgid 5 msg Hello
*

*block 26 msgid 6 msg Hello
*

* *

* *

*block 9 msgid 3 msg PR
*

*block 9 msgid 4 msg rs
*

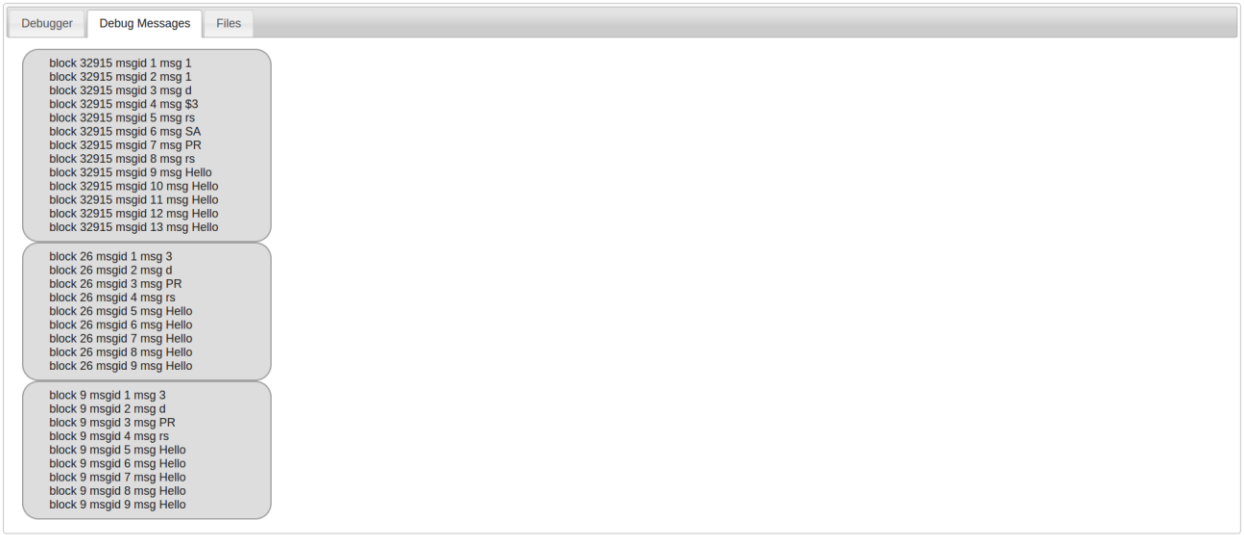
*block 9 msgid 5 msg Hello
*

*block 9 msgid 6 msg Hello
*

* *

* *

The final view of the debugger with the updates logs is shown in the figure below.



Tiny Web Server

File name: tiny.c

Path: oldbb/build/src-bobby/debugger

Description:

The tiny web server is a two threaded http server. The two threads are for

1. Interpreting the commands from the front end by serving the requested content
2. Retrieving messages from the blocks and storing them till they are requested by the front end

All the front end interaction happens in the tiny.c file.

The code for the 2nd thread is in Logger component described below. Functions related to generating messages to the blocks and interpreting messages from the blocks are in this component. This file also contains the code for generating the json objects.

Logger

File name: mylogger.c, mylogger.h, logtable.c, logtable.h

Path: oldbb/build/src-bobby/debugger/logger

Description:

void stringifyLogs(void);

Entry point to the debugger thread. Starts retrieving all the debug messages

void LogTable::stringifyCompleted() ;

Creates JSON object for each message retrieved from the blocks

*void insertLogChunk(Chunk *c) ;*

Interprets the responses from the blocks.

void send_tree_count(void);

For querying the blocks to get the tree count

void ask_attendence(void) ;

To enquire the ids of all the blocks in the system.

Code on the blocks

Filename: name :Log.bb

Path: oldbb/build/src-bobby/system

Description:

This file contains functions for

1. Receiving messages from the server
2. Disseminating the messages throughout the network
3. Interpreting the messages from the server
4. Generating the information
5. Sending the information to the server

The following functions are of interest in this file

void initLogDebug(void);

Starts a spanning tree rooted at the node connected to the host for use by other functions to broadcast messages.

byte handleLogMessage(void);

For forwarding the messages to the host

void commandHandler(void);

For interpreting the messages from host and other blocks

Filename: Span.bb

Path: oldbb/build/src-bobby/system

Description:

Contains the functions to

1) Create spanning tree. The tree can be rooted at the block of choice. The debugger roots the tree at the host PC connection.
2) Broadcasting messages throughout the spanning tree. Callbacks can be broadcast , so that the entire spanning tree can perform a common action
3) Get the count of nodes in the tree
4) Reaching a barrier

Known Issues:

1) Treecount hangs
2) Unpredictable callbacks for neighborhood change events

How to view prints on the debugger

1) Enable debug prints in the Makefile by uncommenting the -DLOG_DEBUG line of the Makefile and run `make wipeout && make`.
2) Use the function printDebug (takes a string as parameter) to send messages to the debugger

How to run the debugger

1) 1)Provide permission to access the blocks using USB with the following commands
    a. sudo chmod 777 /dev/ttyUSB0
    b. sudo stty -F /dev/ttyUSB0 38400
2) Got to the oldbb/build/src-bobby/debugger directory
3) Start the server using the following command

./tiny <port number>

Example : ./tiny 4000

4) Open a web browser
5) Load the front end of the debugger by entering the following url in the address bar of the browser

localhost:<port number>/jquery.html
Example: localhost:4000/jquery.html

6) Click on the Start Logging button on the debugger tab of the front end.
7) Click OK on the alert
8) Go to the Debugger Messages Tab to view the debug messages

Results

Realized a fully functional debugging system for the system of modular robots, capable of displaying state of individual robots in the system.