Development of the Texas Tech Semi Autonomous Language

Kyle Romero

IEEE # 80306278

Texas Tech University

December 7, 2007

Team Members:

Tim Monday

Jacob Smalts

Greg John

Colby Sites

Brian Johnson

Advisor:

Dr. Parten

Abstract

The purpose of this paper is to describe the development of the Texas Tech Semi-Autonomous Language, also known as Tech-SAL.

Tech-SAL is a language designed to be used for robotics applications. Specifically, it is to be used on the Texas Tech ‘Board’ series of microcontrollers, as well as the BEST BRAIN board.

During this development cycle, great strides have been taken to increase the overall usability of Tech-SAL. An integrated development environment has been created and has become the central hub of the Tech-SAL project. Also, an easy to use distribution package has been created to facilitate easier distribution. Lastly, the Tech-SAL language itself has been improved with a greater feature set, with major bugs being corrected at the same time.

Overall, Tech-SAL has become much more useable, and is now approaching a point where it will be able to be used to develop robotics applications easily and effectively.

Table of Contents

List of Figures

List of Tables

Introduction

The *Texas Tech Semi-Autonomous Language (Tech-SAL)* is a high level programming language designed to be used in robotics applications. Specifically, it has been designed to be used with the Texas Tech 'Board' series of microcontrollers, as well as with the BEST Alternate and BRAIN boards.

The original purpose of the creation of Tech-SAL was to spur interest in programming in those who had no formal programming training. To accomplish this, Tech-SAL allows the user to see various levels of programming abstraction, starting from very high level and moving to lower levels. High level programming consists of a programming language that resembles everyday speech and allows the user to perform complicated tasks with relatively few lines of code. Examples of high level programming include the Tech-SAL programming language as well as the C programming language. Low level programming is the type of programming where the code created by the user is very close to the actual code the machine will store in memory. While this type of programming allows great control over what is happening in the machine, it can be difficult to learn, and complicated programming instructions do not exist. Due to this fact, the user must use combinations of simple instructions to perform complicated tasks. The most common lower level programming language is Assembly, which is specific to individual machines. Figure 1 is a flowchart representing the compilation / translation process flow of the system.

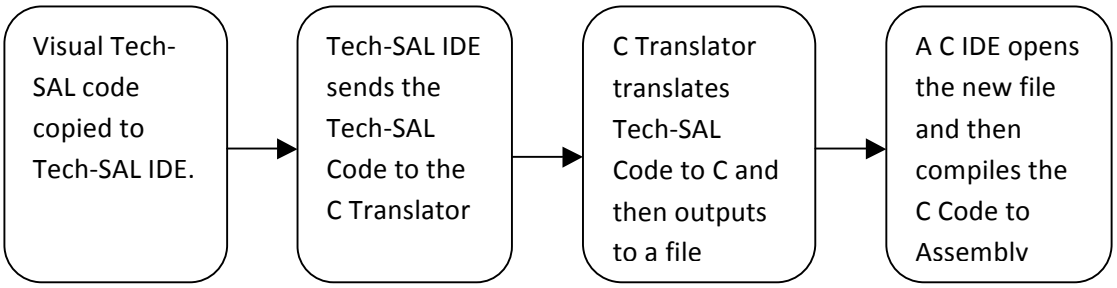


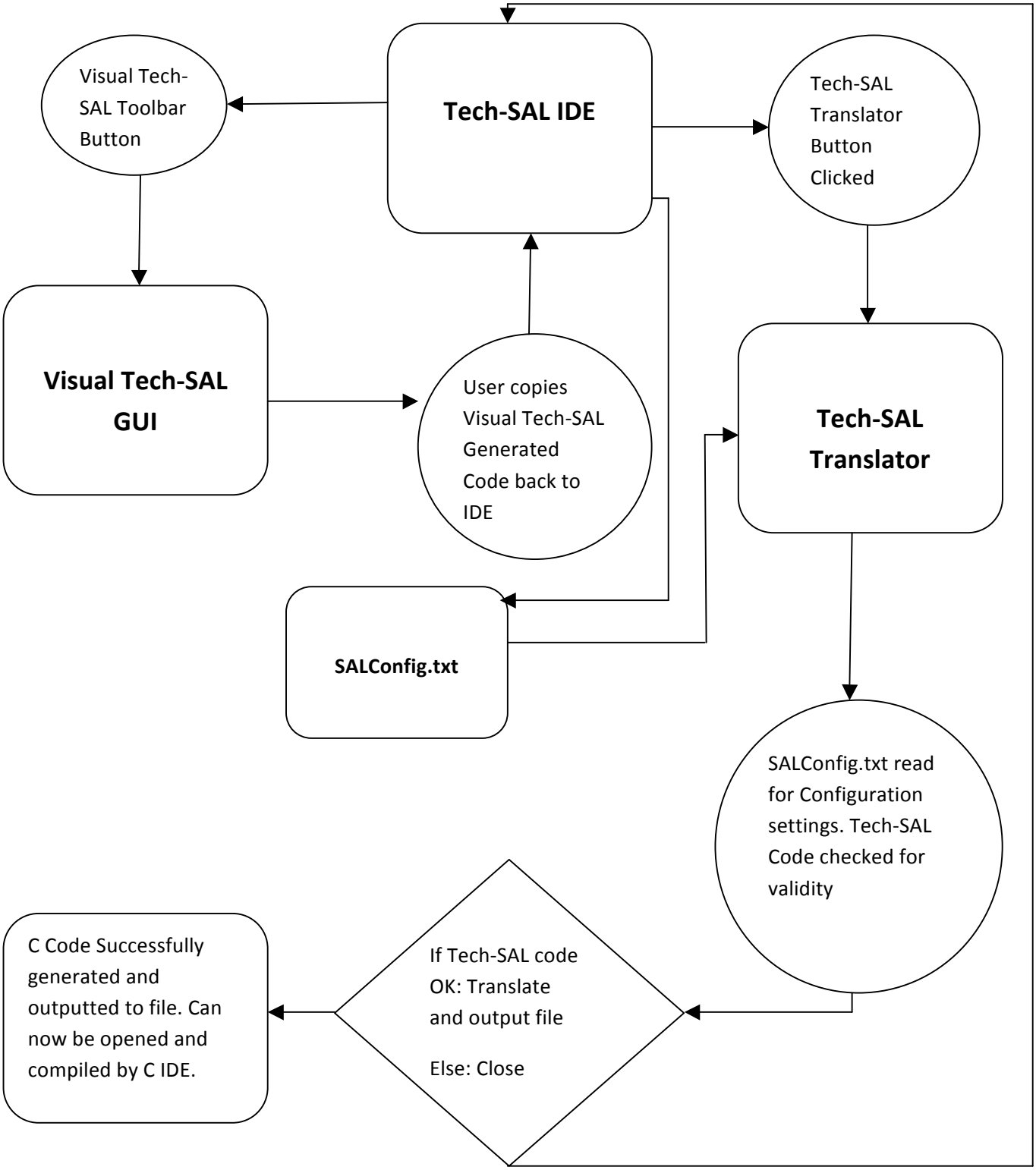
Figure 1- The Tech-SAL Compilation Process

As can be seen from figure 1, as one moves from the left box to the right box, one moves from a high level programming level to a gradually lower one. It is important to note that the three left boxes are part of the Tech-SAL project; the right box requires a separate program that is not being developed by Texas Tech. Currently, the Tech-SAL C translator produces code for the two following C integrated development environments: IAR Workbench and CCE.

Tech-SAL Components

Before describing each component of the Tech-SAL project in detail, visualizing an overall view of the system will allow a better understanding of where each component lies in the Tech-SAL'process flow. Figure 2 will visualize the connections between the different components.

Figure 2- Tech-SAL Component Interconnections



The Distribution Package

The Tech-SAL distribution package is a convenient subject to start the discussion of the Tech-SAL components due to the fact that it is the first thing the user will encounter while dealing with Tech-SAL. The distribution package is a Microsoft installation file (.msi), and mimics the standard Microsoft Windows installation structure. Figure 2 is a screenshot of the distribution package's installation screen.

Figure 3- Distribution Package

The main reasoning behind the creation of the distribution package was to maximize the ease of use for the end-user. The installation package was created using Microsoft Visual Studio 2005. It includes all necessary file required to run the different components of Tech-SAL, and uses a pre-designed directory structure to ensure all components of the project operate correctly. The package also registers the '.tsal' file extension, which is used on Tech-SAL code files. When a file with the '.tsal' file extension is double clicked, it is automatically opened in the IDE. While this part of the project is limited in technical aspects, it is nonetheless an important inclusion in the project because any modern piece of software should include a good installation

program. The average user will be quite displeased with a piece of software that does not automatically set-up all components of the software automatically.

The Integrated Development Environment

The Tech-SAL Integrated Development Environment (IDE) is a recent addition to the Tech-SAL project. It was designed as a way to interconnect all the separate components of the project into a central hub.

Currently, the IDE is at an early stage of development. Already, however, it is becoming an increasingly important part of the project. Figure 3 shows a screen shot of the IDE.

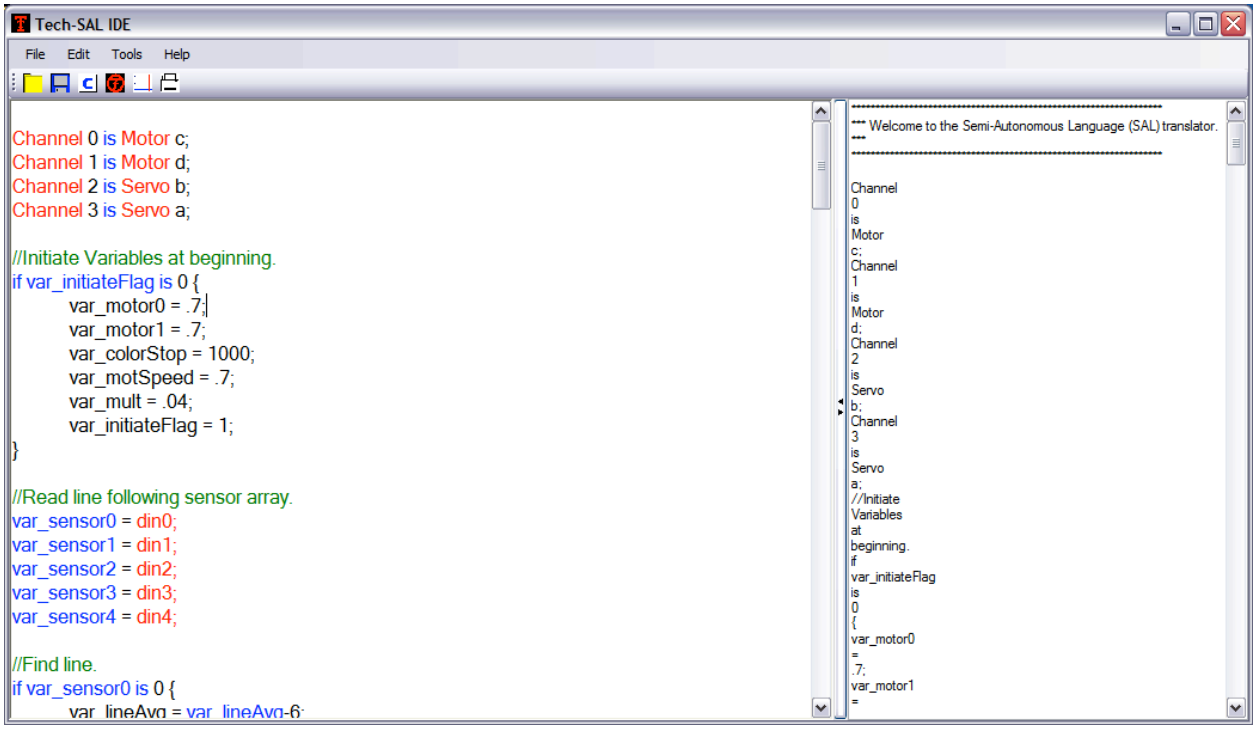


Figure 4- Tech-SAL IDE

The main field of the IDE is a text box. This text box is where Tech-SAL code can be inputted. One nice feature of this code input area is the inclusion of real-time syntax highlighting. Any keyword defined by the Tech-SAL language syntax will automatically be highlighted. This allows for easier reading and debugging of code.

To the right of the main input field is the output of the Tech-SAL C translator. This output informs the user of any errors that occurred during the translation process from Tech-SAL to C. Above the text box is the main toolbar of the IDE, where the IDE links to other components of the project. To The toolbar allows commonly used functions to be called quickly. Currently, the toolbar contains the following options:

- Open
- Save
- C Translator
- Visual Tech-SAL GUI
- SALConfig.txt
- Print

The Open / Save buttons, as their names imply, provide the ability to load a saved Tech-SAL '.tsal' code file into the text box or to output the current contents of the text box to a .txt file. The IDE uses the standard Windows Save As / Open dialog boxes.

The C Translator Button links to the Tech-SAL Translator executable. To get the translator to work correctly with the IDE, the translator had to be modified to accept run time arguments. The IDE passes in two arguments to the Tech-SAL Translator when this button is selected, the location of the input file, which is always a temporary .tsal file named "temporarytechsalfile.tsal", and the location of the output file, which is selected from a Save As dialog window.

The reason for always using the temporary input file is to ensure the current code in the IDE text field gets sent to the translator. This file is generated by outputting all contents of the text box to this text file.

Also, one major improvement that the IDE has made to the translation process is the fact that the IDE will allow you to output the file to any location that is desired. Before the implementation of the IDE, the translator would only output the file to its home directory. The process of saving the output file to any location is accomplished by first outputting the file to the translator's home directory and then copying it over to the desired location. After the file has been successfully copied, the original is deleted.

The SALConfig.txt button causes a secondary window to appear on the screen. This window allows one to customize the SALConfig.txt file. This file is used by the Tech-SAL translator to control various translation options, such as the target environment (IAR or CCE) to output the code for. Figure 4 is a screenshot of this window.

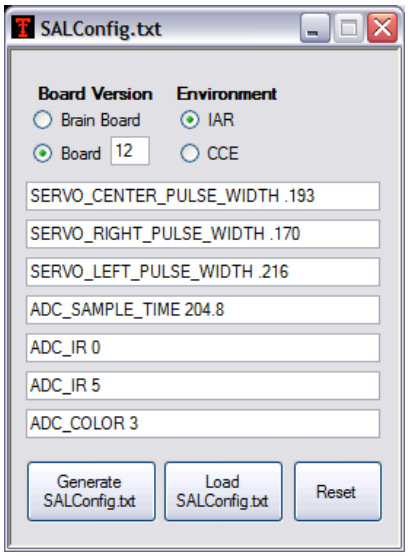


Figure 5- SALConfig.txt

As can be seen, this window provides options to modify the current microcontroller being used, the environment to output to, as well as other miscellaneous options. It also has buttons to allow you to load a current SALConfig.txt file or to output the contents of the window to the correct location on the hard drive to be used by the Tech-SAL Translator. The values in the table default to sane values, and the

SALConfig.txt file will be auto-generated if no file is found in the Tech-SAL Translator's home directory. The auto-generation of the SALConfig.txt file is to prevent a crash while translating in the event that no SALConfig.txt file is located.

The Visual Tech-SAL button allows one to call the Visual Tech-SAL GUI. This GUI allows the visual creation of Tech-SAL Code using graphical icons and drag and drop techniques. The process of using the Visual Tech-SAL GUI will be covered more in following sections.

Finally, there are the file menus at the top of the IDE window. These contain common options, such as editing (Cut, copy, paste, etc.), help (Readme, Quickstart guide), and also mirror links to the functions in the toolbar (the tools menu). Also, several unimplemented features, such as the update tool and graphical boot strap loader are linked to in the tools menu. The features, which may prove useful in the future, are currently included to simply show what could be done with the IDE in the future.

Technical Aspects of the Tech-SAL IDE

For the benefit of the program developer, several important aspects of the Tech-SAL IDE must be explored.

For starters, it was written in the C# programming language. As a result of this fact, Tech-SAL IDE is a Windows only program for the time being, since C# relies heavily upon the Windows API. The IDE was developed using Microsoft Visual Studio 2005, and was creating using a Windows Form Project.

Most of the Tech-SAL IDE is event handling or the Windows Form design. While the graphical aspects of the Windows Form were created by the developer, the underlying code of each Form component was auto-generated by Visual Studio. The developer mainly focused on creating what the user will see, as well as creating event handling routines to control what will happen when various controls are activated.

All of the Open / Save dialog windows were created using the Standard Microsoft Windows forms. This fact allowed a familiar touch while using the software. Also, special care was taken while coding the Open / Save dialogs to keep track of where in the directory structure of the hard drive the user was currently located. Had this special care not been taken, the IDE would crash due to the inability to locate required components, such as the Tech-SAL Translator, and would not be able to Save/Open files correctly, due to the fact that the program would think it was located in a different directory than it was in reality.

All icon files were created using Microsoft Visual Studio 2005 and were saved in the standard icon format (.ico). Also, the main dependency of the Tech-SAL IDE, besides requiring the other components of the Tech-SAL Project, is the .NET Framework 2.0. If this file is not located on the host computer, then the IDE will fail to run.The technical aspects of the IDE are summarized in Table 1:

Table 1- Tech-SAL IDE Technical Details

Language	C#
Files	Form1.cs Form1.Designer.cs Form2.cs Form2.Designer.cs Program.cs Icon1.ico Config.bmp Compile.bmp Save.bmp Visual.bmp
Requirements	.NET Framework 2.0

Tech-SAL Translator

The aforementioned Tech-SAL translator is the program that converts Tech-SAL Code into C Code. This is a standalone program that has itself been written in the C language. Its main function is to accept the location of a Tech-SAL .txt file and then convert its contents into the C programming language. Figure 5 shows a screen shot of the translator converting an example block of Tech-SAL code.

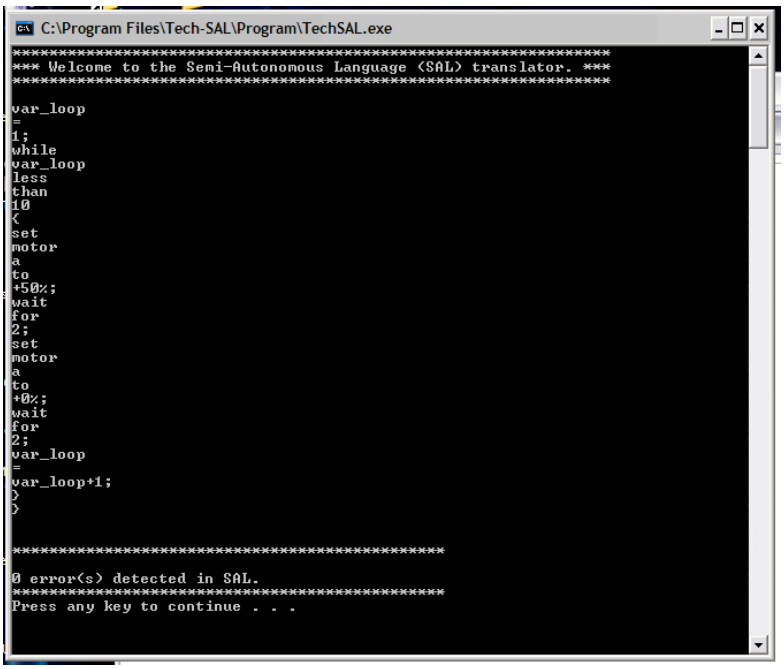


Figure 6- Tech-SAL Translator

The translator includes several important features. The most important feature of course is the actual translation process. The translator steps through the inputted text and picks out Tech-SAL keywords. Then the translator looks at its SALConfig.txt file and determines which environment and board it needs to target. Using options defined in the *SALConfig.txt* file, the translator then passes the code through a syntax checker. This checker stills requires much more work, but at the moment currently operates well enough for general use. If the syntax checker detects any incorrect Tech-SAL code, the translator halts translation. Otherwise, the translation finishes and is outputted the file to the user defined location. The only major modification required, as previously mentioned, was the addition of run time parameter checking to allow compatibility with the IDE.

Currently, the Tech-SAL Translator can generate valid code for all of the Texas Tech Board series up to Board 12. It also included generation capabilities for the West

BEST Alternate and BRAIN boards. One thing to note is that the “SALConfig.txt” file is significantly different for Tech Boards or BEST Boards. The Tech boards require fewer options, and the window in the IDE will automatically blank out components that are not required in the event that the Board series is the selected board. The IAR workbench is currently fully supported, while CCE operability is currently in development.

After the translator has finished translation to C, the C file must then be loaded into one of the previously mentioned C IDEs. The file can then be modified to suit the user’s needs, and compiled within the C IDE and sent to the appropriate micro-processor. All of the currently supported board architectures use the MSP430x series of microprocessor.

One important final note about the Tech-SAL C translator is the fact that its output is redirected to the IDE. In previous versions of the Tech-SAL project, the translator would be called as a standalone program. Now, however, it is called from within the IDE. The user never sees the translator window. Instead, the output is sent to the aforementioned text box in the IDE. The redirection of the output allows for a more uniform and solid feel to the IDE.

The Visual Tech-SAL GUI

The Visual Tech-SAL GUI, mentioned in the IDE section, is a visual means of producing Tech-SAL Code. Figure 6 shows this GUI.

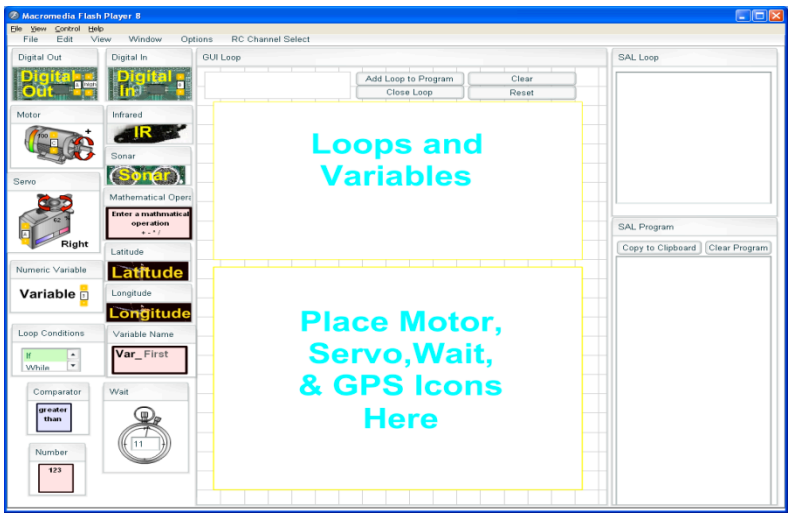


Figure 7 [2]- Visual Tech-SAL GUI

The program was written in Macromedia Flash to allow a visually pleasing user interface.

The column on the left contains the different options for creating the code, and they must be dragged to the center column to create the valid code. The order in which they are added determines the order of the Tech-SAL code generated in the right column.

One important modification made in the current development cycle is the addition of a switch version option in the File menu. This option allows the user to switch to a more basic version of the GUI, with much fewer options. This new feature is intended to make the code creation process simpler for users that do not require advanced functionality, such as GPS or Infrared sensor input.

Currently, the Visual Tech-SAL GUI does not have an implementation for Text output or input. Due to this fact, to get the generated code into the Tech-SAL IDE, the user has one of two options:

1.) Copy the generated code into notepad and then save the file. Once the file is saved, open it with the IDE.
2.) Copy the generated code directly into the IDE text box.

Language Capabilities

In addition to the components of the Tech-SAL project, the Tech-Semi Autonomous Language itself contains many features that must be described.

The language has the useful ability to directly set and control motors, servos, and digital I/O pins, which is useful because it allows easy manipulation of peripheral devices connected to a board using Tech-SAL. In addition, the language can be used to set RC channel assignment quite easily, allowing the board's peripheral motors and servos to be controlled by a RC control unit.

The language has the ability to accept input from various input devices, such as infrared, GPS, and sonar sensors. These sensors can be set as variables, and used during program execution to determine their current state.

The language also contains conditional and loop structures, which are common to many programming languages. The conditional statements supported by Tech-SAL are:

- If
- Is
- Is not
- And

- Or
- Greater than
- Less than

The looping structures supported are:

- While
- Wait
- For

Additionally, the language supports Single and Multi-Line commenting, which allows Tech-SAL code to be passed from one programmer to another quite easily because of the inclusion of built in comments. A full listing of the Tech-SAL Syntax with examples is included in Appendix A. This data was taken from the Tech-SAL Readme.doc file.

One final capability of the language currently in development is differential steering. This feature allows the mapping of two motors or servos to one channel, which allows for in-place steering using two motors instead of directional steering using a servo.

Future Development

This project is still in active development and will remain so for quite some time. There are several important future developments that are desired for this project.

First of all, the Tech-SAL IDE will require much more development to get out of its current Alpha level status. While currently the IDE is at a usable state, it is still

relatively simple, and will require much more functionality, such as built in syntax checking and internal C translation, to make it a professional program.

Also, an update procedure is currently in the works. This would allow the software to query an online server to check if Tech-SAL is currently the most updated version. If not, then the updated files could be downloaded into the program. At the current moment, the update tool is implemented in the IDE, but is not used due to no update server being set up. The client side software is show in Figure 8:

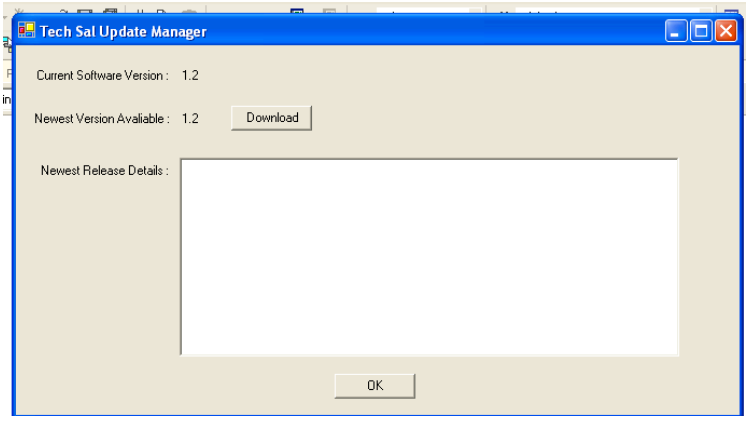


Figure 8 [2] - Tech-SAL Update Manager

At the same time as Tech-SAL is being developed, Tech's Board series is constantly being updated, and as such, the Tech-SAL project will have to maintain operability with the Board series by adding header files for every new iteration.

Finally, one major improvement desired in the future is the ability to compile correctly with both the IAR Workbench and CCE. That is currently being heavily developed.

Conclusion

This development cycle of the Texas Tech Semi Autonomous Language has seen quite an increase in usability. The goal of this group has been to allow the user to operate all components of the project from one central hub, the Tech-SAL IDE, as well as increasing ease of use in general.

The installation package has been quite a nice addition to the project, and allows much easier distribution of the Tech-SAL software. In previous iterations, a person would have to search through a jumble of files contained in a .zip file to get Tech-SAL operational. Now one must only navigate to the Start menu in Windows.

The Tech-SAL IDE has become quite operational, and should make the Tech-SAL project and programming language much more enticing to people in the near future. Whereas before, a user would have to be technically knowledgeable to operate Tech-SAL, now anyone with any sort of computer knowledge will be able to use Tech-SAL to great effect.

Finally, the Tech-SAL language itself has seen quite an increase in functionality. Many of the major bugs experienced in previous versions of the software have been corrected, and several new features, such as differential steering, have been added. All-in-all, Tech-SAL is approaching the point of being a usable and useful piece of software for the development of robotics applications. Tech-SAL should live up to its main goal of getting people interested in programming even with no technical background.

Works Cited

1.) Tech-SAL Development Groups, “Tech-SAL Readme, v7.0.doc”, Pgs. 5-11, 10/21/2007
2.) Jacob Smalts, “EE3331S01P01 Interim Presentation”, Slides 20 and 22, 10/21/2007
3.) Greg John, “EE3331S01P01 Interim Presentation”, Gantt Charts, 10/21/2007
4.) Timothy Monday, “EE3331S01P01 Interim Presentation”, Budget, 10/21/2007

Appendix A: Tech-SAL Language Syntax [1]

4. LIST OF COMMANDS

The following is a list of commands that are currently usable in Tech-SAL, and a brief description of how to use them.

4.1 set motor

Used to set the current motor to a specific speed. You can use variables or constants. Use + on a variable in order to have the variable's sign determine positive or negative.

Ex: set motor A to +var_motorspeed%;

Ex: set motor b to +50%;.

4.2 move servo

Used to change the direction that a specific servo is turned. Use + on a variable in order to have the variable's sign determine positive or negative. A negative value moves the servo to the left that percentage, a positive value moves the servo to the right that percentage.

Ex: set servo B to -54%;

Ex: set servo A to +var_servA%;

4.3 if/while

Used as a conditional statement for a robot to make decisions. The exact same statement can be made with a while loop.

Ex: if var_1 greater than 8 {

set motor A to +100%;

}

Ex: while var_1 greater than 8 {

Led;

}

4.4 is/is not

Used as comparator for values. "is" compares two values for equality, "is not" compares two values for inequality.

Ex: if var_1 is not 8 {

Led;

}

Ex: while var_2 is 80 {

Led;

}

4.5 and

And is used two test two variables at the same time.

Ex: if var_1 and var_2 less than var_3 {

Set servo A to +var_servAvalue%;

}

4.6 or

Or is used to test two variables

Ex: if var_1 or var_2 less than var_3 {

Set servo A to +var_servAvalue%;

}

4.7 wait

Wait for a specified amount of milliseconds.

Ex: wait for 1450;

4.8 for

for is used to specify a loop, when the starting and ending conditions are known, as well as an increment.

Ex: for var_2 = 2. var_2 less than 3. var_2++. {

set motor A to +30%;

}

4.9 Single Line Comments

You can add comments to your code with the // command. This will make any other words on the same line non-functional.

NOTE: The double forward slashes must be separated on either side by a white space. And the comment must end with a '.'

Ex: set motor A to +100%; // Turn motor A on to full power.

4.10 Multiple Line Comments

You can add comments to your code with the /* */ command. This will make any other words between these two symbols non functional.

NOTE: Each line of the comment must end with a '.'

Ex: /* This is an example.

Of a multiple line comment */

4.11 variables

The user has dynamic variables that they may assign a value.

These variables are all of the "double" type.

Ex: var_myvar = 4.5;

4.12 greater than / less than operators

Used to compare two values.

Ex: if var_myvar greater than 1 {

var_1 = 10;

}

Ex: if var_myvar less than 1 {

var_1 = 10;

}

4.13 expressions

Any math can be done in C can be done in Tech-SAL.

NOTE: everything after the equals sign is dumped directly into C, therefore it must be valid C code or you will have errors compiling it in IAR. Any math operators used after the equals sign must be valid C operators and be defined in math.h

NOTE: one space after the equals sign there can be no other spaces or it will not translate properly to valid C code.

Ex: var_latdistance = var_lat2-var_lat1;

Ex: var_myvar = var_3*2;

Ex: var_yourvar = cos(var_3)*var_myvar;

4.14 include

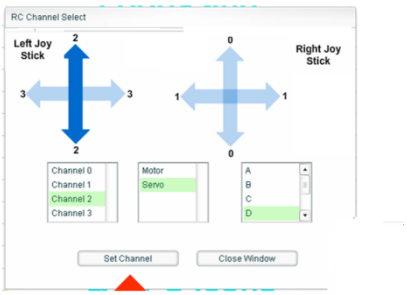
Multiple separate files can be included in the main Tech-SAL file by using the include command. When the translator sees an include, it takes the code in that text file and places it where the include was read.

Ex: include sal2.txt

NOTE: included files cannot include more files.

4.15 RC Channel Selection

RC channels can now be selected to move a certain servo or motor. Channels can ONLY be defined at the beginning of the SAL file (shouldn't need anything else if you are using RC).



1-3 corresponds to RC channels, 4 is OFF

Ex: Channel 2 is Servo D;

Ex: Channel 1 is Motor A;

5. INPUT & OUTPUT

5.1 IR

Setting a user defined variable equal to IR1 will read the voltage from the first IR sensor declared in the config file. IR2 will read the voltage from the second IR sensor declared in the config file and so on. This variable can then be used in other Tech-SAL commands.

Ex: Var_ir1value = IR1;

```
If var_ir1value greater than 2 {
        Set servo A to +20%;
}
```

5.2 GPS

The GPS unit connected through a transceiver will be connected to pin33 of the MSP for USART input. In the SAL_config.txt file you will need to set the baud rate of the GPS unit you are using as shown above.

There are two global variables you can use at any time in Tech-SAL that hold GPS data. "latitude" holds the GPS latitude, and "longitude" holds the gps longitude.

These variables will fetch the most current location data from the Garmin GPS when called which updates at 1Hz (once a second).

Example longitude return – 12345.6789

Example latitude return – 1234.5678

Ex: var_mylat = latitude;

var_mylong = longitude;

5.3 GPS Valid

The global variable gps_valid is available to the user to check if the Garmin GPS is acquiring valid GPS data. This variable is initialized to zero and set to one when the global variables longitude and latitude were successfully filled with valid data. To check, a used defined variable must be set equal to gps_valid, that variable can then be used in other commands.

Ex: var_loop = 1;

```
while var_loop is 1 {
        var_mylat = latitude;
        var_mylong = longitude;
        var_gpsvalid = gps_valid;
        If var_gpsvalid is 1 {
        //Do stuff
        }
}
```

5.4 Sonar

A call to sonar will return the distance that your sonar device currently reads in centimeters. For board 7 & 8 sonar must be connected as such- port 1.1 (pin 13) to receive, and port 1.4 (pin 16) to trigger.

Ex: var_sonardist = sonar;

5.5 Digital In

Setting a user defined variable equal to din0 or din1 stores the current state of that digital input (1 or 0). This variable can then be used in other Tech-SAL commands. Checking Digital In can also be done in

Ex: Var_digizero = din0;

If var_digizero is not 1 {

Set servo A +20%;

}

5.6 Digital Out

Changing the state of the digital out ports is done as such. Digital out 0 and 1 on your board are called as “dout a” and “dout b”.

Ex: set dout a to low;

Ex: set dout a to high;

Ex: set dout b to low;

Ex: set dout b to high;

5.7 H-Bridge Status

When the H-bridge status input pin of the MSP reads a falling edge, the PWM to the H-bridge is terminated and the MSP’s LED is lit. See your board user’s manual for what port to connect the status input to.

Setting a user defined variable equal to hbridge_fail stores the current state of the H-bridge. This will be 1 if the H-bridge has failed and the PWM was disabled.

NOTE: Contrary to the board 7 & 8 user’s manual, this input has been moved to port 1.6 (pin18)

IMPORTANT: This interrupt triggers on receiving a falling edge on the H-bridge status port. If the LED lights up on your MSP then the MSP has terminated the H-bridge PWM due to this interrupt. You must not let this port float.

Appendix B: Gantt Charts [3]

WEEK 1	8/28/07	9/4/07
Review Previous Final Reports	8/28/07	9/4/07
Consult Previous Project Programmers	8/28/07	9/4/07
Review Current SAL Translator	8/28/07	9/4/07
Review Current SAL Syntax Detector	8/28/07	9/4/07
Review CCE Compatibilty	8/28/07	9/4/07
Review Current Documentation	8/28/07	9/4/07
Review Current GUI	8/28/07	9/4/07
WEEK 2	9/4/07	9/11/07
Electronic Scoring Setup (Hardware)	9/4/07	9/11/07
Electronic Scoring Setup (Software)	9/4/07	9/11/07
Fix BEST Demo Robot for Kick-Off	9/4/07	9/11/07
Revise CCE and RC Code	9/4/07	9/11/07
Revise SAL Translation Code	9/4/07	9/11/07
Revise GUI Code	9/4/07	9/11/07
WEEK 5	9/25/07	10/2/07
Add Control Set to GUI (Brain, Board 12)	9/25/07	10/2/07
ReWrite Motor (B12), header files	9/25/07	10/2/07
Change 'period' to 'semi-colon' for SAL...	9/25/07	10/2/07
Differential Steering Mapping w/ Brain ...	9/25/07	10/2/07
Finish CCE Code	9/25/07	10/2/07
Modify 'for' loop statement code	9/25/07	10/2/07
WEEK 6	10/2/07	10/9/07
Finalize SAL Package	10/2/07	10/9/07
Finish Board Header Files	10/2/07	10/9/07
Add Save/Open Function to GUI	10/2/07	10/9/07
Acquire Scoring Setup (Projector, Lapt...	10/3/07	10/6/07
Continue Differential Steering Mappin...	10/2/07	10/9/07
Board Testing	10/2/07	10/9/07
Draft SAL commenting and code conv...	10/2/07	10/9/07

WEEK 3	9/11/07	9/18/07
Tested RC Code from SAL and BEST	9/11/07	9/18/07
Create GUI User Documentation	9/11/07	9/18/07
Review Schematics for Board 12 Moto...	9/11/07	9/18/07
Create BEST GUI Version	9/11/07	9/18/07
Begin modifying SAL to translate RC C...	9/11/07	9/18/07
Revise CCE and RC Code	9/11/07	9/18/07
WEEK 4	9/18/07	9/25/07
Create SAL Package	9/18/07	9/25/07
Create Board 12 Motor and Servo Code	9/18/07	9/25/07
Revise SAL Tranlation for BRAIN board	9/18/07	9/25/07
Create GUI Versions (BEST, SAL)	9/18/07	9/25/07
Test Motor/Servo Outputs for Brain B...	9/18/07	9/25/07
Revise CCE Code	9/18/07	9/25/07
WEEK 7	10/9/07	10/16/07
Complete SAL IDE	10/9/07	10/16/07
Board 12 H-Bridge Interrupt	10/9/07	10/16/07
Revise/Test GUI Code	10/9/07	10/16/07
Finish BRAIN RC Differential Mapping	10/9/07	10/16/07
Revise/Test CCE Code	10/9/07	10/16/07
Continue SAL commenting and code re...	10/9/07	10/16/07
Prepare Interim Presentation and Rep...	10/9/07	10/16/07
WEEK 8	10/16/07	10/23/07
Revise SAL Package	10/16/07	10/23/07
SAL Software Support	10/16/07	10/23/07
Revise/Test GUI Code	10/16/07	10/23/07
Acquire Scoring Setup (Project, Lapto...	10/18/07	10/20/07
Revise/Test SAL Translation Code	10/16/07	10/23/07
Revise/Test CCE Code	10/16/07	10/23/07
Compile and Upload New Version	10/16/07	10/23/07

WEEK 9	10/23/07	10/30/07
Revise Documentation	10/23/07	10/30/07
Test Software	10/23/07	10/30/07
Revise GUI Code	10/23/07	10/30/07
Revise SAL Translation Code	10/23/07	10/30/07
Revise CCE Code	10/23/07	10/30/07
SAL Software Support	10/23/07	10/30/07
WEEK 10	10/30/07	11/6/07
SAL Software Support	10/30/07	11/6/07
Revise Documentation	10/30/07	11/6/07
Test GUI Code	10/30/07	11/6/07
Test SAL Translation Code	10/30/07	11/6/07
Test CCE Code	10/30/07	11/6/07
Compile and Upload New Version	10/30/07	11/6/07
WEEK 11	11/6/07	11/13/07
Revise Documentation	11/6/07	11/13/07
Test Software	11/6/07	11/13/07
Reviset GUI Code	11/6/07	11/13/07
Revise SAL Translation Code	11/6/07	11/13/07
Revise CCE Code	11/6/07	11/13/07
SAL Software Support	11/6/07	11/13/07
WEEK 12	11/13/07	11/20/07
SAL Software Support	11/13/07	11/20/07
Revise Documentation	11/13/07	11/20/07
Test GUI Code	11/13/07	11/20/07
Test SAL Translation Code	11/13/07	11/20/07
Test CCE Code	11/13/07	11/20/07
Compile and Upload New Version	11/13/07	11/20/07

Appendix C: Budget [4]

Projected Budget

Estimated Budget	(09/03/2007)			
Labor Costs				
Name	**Hourly Pay Rate**	**Hours Worked**	**Total Pay**	
Colby Sites	$15.00	210	$3,150.00	
Greg John	$15.00	210	$3,150.00	
Brian Johnson	$15.00	210	$3,150.00	
Kyle Romero	$15.00	210	$3,150.00	
Jacob Smalts	$15.00	210	$3,150.00	
Tim Monday	$15.00	210	$3,150.00	
		Total Labor Cost	$18,900.00	
		75% Overhead	$14,175.00	
		Total Labor and Overhead	$33,075.00	
Parts Costs				
Part Name	**Cost**	**Quantity**	**Total Cost**	
Misc. Parts	$50.00	NA	$0.00	
		Total Parts Cost	$0.00	
Equipment Rental Costs				
Equipment Name	**Purchase Cost**	**Day Rental Rate**	**Days Rented**	**Total Cost**
BEST "Brain" Control Board	$200.00	$0.40	98	$39.20
Programming Cable	$100.00	$0.20	98	$19.60
Multimeter	$72.80	$0.15	98	$14.27
"Brain" Board Remote Control	$200.00	$0.40	98	$39.20
Oscilliscope	$1,377.70	$2.76	98	$270.03
Unknown	$1,000.00	$2.00	98	$196.00
			Total Rental Costs	$578.30
Total Estimated Budget	**$33,653.30**			

Tim Monday (10-15)

Appendix D: Evaluation

WRITTEN LAB REPORT EVALUATION FORM

Student Name: Course Number:

Please score the student by circling one of the responses following each of the statements.

1) The student's writing style (clarity, directness, grammar, spelling, style, format, etc)

A B C D F Zero

2) The quality and level of technical content of the student's report

A B C D F Zero

3) The quality of results and conclusions

A B C D F Zero

4) Quality of measurements planned/ taken

A B C D F Zero

Grade: