Clips To C Interface
For Mobile Robot Control

Graduate Project Final Report

Submitted By
Sean Babcock

To The
Department Of Computing and Mathematical Sciences
Texas A&M University – Corpus Christi

Fall 2000

Committee Members

R. Stephen Dannelly, Ph. D.
Chairperson

Mario Garcia, Ph. D.
Committee Member

Michelle Moore, Ph. D.
Committee Member
Abstract

This project is the design and implementation of a series of library functions written in the C programming language that are used to extend the functionality of CLIPS, an expert system shell. Students enrolled in an expert systems class will have the ability to write CLIPS programs that utilize the C libraries to control a Pioneer Mobile Robot. The function calls return a series of values back to the CLIPS expert system inference engine denoting the current state of the robot. Previously, all robot control programs had to be written in C. With this project, programs can now be written in a higher level language, CLIPS.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>ii</td>
</tr>
<tr>
<td>List of Tables and Appendices</td>
<td>iii</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>II. Background</td>
<td>3</td>
</tr>
<tr>
<td>III. Project Tasks</td>
<td>8</td>
</tr>
<tr>
<td>IV. CLIPS Interface Functions</td>
<td>10</td>
</tr>
<tr>
<td>V. Conclusion</td>
<td>19</td>
</tr>
<tr>
<td>Bibliography</td>
<td>21</td>
</tr>
</tbody>
</table>
List of Tables and Appendices

Table 1. Port types and Saphira-defined names for client/server connections. Page 5
Table 2. Connection example. Connecting a Pioneer Application Interface Client with a Robot. Page 5
Table 3. Disconnecting example. Disconnecting a PAI Client from a Robot. Page 6
Table 4. Connect/Disconnect example. PAI connecting/disconnecting with a Robot. Page 6
Table 5. DefineFunction in PAI. Page 9
Table 6. UserFunctions in PAI. Page 9
Table 7. Interface functions. Functions that will be available for use within a CLIPS application. Page 11

Appendix 1. StopRobot.c function source code. Page 22
Appendix 4. StartupRobot.c function source code. Page 25
Appendix 5. ShutdownRobot.c function source code. Page 27
Appendix 7. ControlRobot.clp CLIPS expert-system source code. Page 29
I. Introduction

Artificial-intelligence languages were first introduced as having the ability to imitate the human brain, however, creators were disappointed to realize that the human brain was more complex than previously considered [2]. Current research on artificial intelligence has led to programming languages like CLIPS. CLIPS is an expert-systems programming language developed and currently used by NASA. CLIPS is currently used in large corporations for routing documents and allowing for delay factors [3], as well as in industrial environments as a tool for complex process control [5].

Up to this point, students enrolled in the Expert Systems class at the Texas A&M University—Corpus Christi (TAMU-CC) Computer Science Department were not receiving experience programming robots. The department's mobile robots available to students are programmed in C, which is not conducive to expert-systems programming. The goal of this project was to expand the capabilities of the course by allowing the students to use the expert system language known as CLIPS, or C Language Integrated Production System, by creating a series of interface functions to control a Pioneer Mobile Robot.

Allowing students to learn another language, especially one which is of such a growing importance to our industry, is of great benefit. TAMU-CC students and staff will benefit by enhancing their knowledge, experience, and marketability.

Students in an Expert Systems class will now be able to complete projects such as programming a robot to traverse a path containing obstacles while utilizing expert systems features of CLIPS. The students will write expert system code to include what to do when an object is blocking the path of the robot. Another project could include
mapping a room by allowing the robot to roam while keeping track of objects in the room. There are many possibilities that can become exciting expert systems projects.

The task of interfacing CLIPS programs with C programs was a challenging project requiring knowledge of C as well as learning a unique language, CLIPS. This is an exciting opportunity for a graduate student to provide a new learning tool for all students to follow.

Chapter 2 of this document, Background, overviews the use of both CLIPS and the ActivMedia Mobile Robot Program Application Interface. Chapter 3 of this document, Project Tasks, describes the design features of CLIPS that allow the use of the new interface functions. Chapter 4, CLIPS Interface Functions, illustrates the functionality of each of the new interface functions. Chapter 5 of this document, Conclusion, summarizes the results that were attained and suggests future work to be considered.
2.1 CLIPS

CLIPS (C Language Integrated Production System) is a tool developed by the Software Technology Branch (STB), NASA/Lyndon B. Johnson Space Center for writing applications called expert systems. An expert system is a program that is specifically intended to model human expertise or knowledge [6]. An expert system contains data, called facts, and heuristics, which are the rules that relate to the data. Together, the rules and the facts form a “knowledge base”.

“A production-system program consists of an unordered collection of IF-THEN statements called productions. The data operated on by productions is held in a global database called working memory” [4]. The inference engine, during execution of the production system, performs “match”, “conflict resolution”, “act”, and “repeat” operations. The “match” operation performs an evaluation of the current contents of working memory with the left side of the production. The “conflict resolution” operation selects the production that satisfies the left side. The “act” operation performs the production’s right-side action. The “repeat” operation simply repeats the previous operation [4].

CLIPS was designed to allow the two kinds of integration that are necessary when designing an expert-system allowing the capability to call external functions from CLIPS and embedding CLIPS in other expert systems [6]. The integration that is used in this project is the capability to call external functions from CLIPS. The external functions are written in the C programming language and are called from within a CLIPS application. When the execution of an external function completes, control returns to the CLIPS
inference engine. “The easy addition of external functions allows CLIPS to be extended or customized in almost any way” [6].

The main goal of this project was to extend the capability of CLIPS to allow a CLIPS application to control a Pioneer Mobile Robot. Since the Pioneer Mobile Robots are programmed in C, as described in Section 2.2, some type of interface to allow a CLIPS application to control the robot was needed. The interface developed by this project is accessed by means of function calls from within a CLIPS application. Hence, student programmers will not be burdened with the task of designing algorithms to control the robot's sonars and motors. They have the capability to simply make a call to a C function while focusing their efforts on developing heuristics appropriate to the problem at hand. The interface functions have the ability to detect an error and return an error code back to the calling CLIPS routine for further error handling.

2.2 Saphira API

The department's Pioneer Mobile Robots were previously only controlled with programs written in C using Saphira. Saphira is the C language-based, robotics-application development environment created by SRI International. To make the task of controlling the robots with Saphira easier, a series of C library functions were created. These C library functions were designed to control the robots from the C language and are collectively called the PAI or Pioneer Application Interface. The PAI is a collection of API’s, Application Program Interface routines, which allow the programmer to easily control the robot without dealing with low-level robot control programming. Programs written on a Unix system with Saphira are compiled using either gcc or cc compilers [6].
The Pioneer Mobile Robots are controlled in a client-server environment. The first thing the PAI client must accomplish is to connect to the robot server. The PAI function paiRobotStartup makes this connection. Three different methods exist to make this connection. A client/server connection can be established either through a serial port, a TCP/IP network or to a simulator server on the same machine [6]. Table one obtained from the PAI Programmers Manual [1] describes these methods.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SfLOCALPORT</td>
<td>Connect to simulator on the host machine</td>
</tr>
<tr>
<td>SfTTYPORT</td>
<td>Connect to robot on tty port</td>
</tr>
<tr>
<td>SfTCP PORT</td>
<td>Connect to robot over TCP/IP network</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SfCOMLOCAL</td>
<td>Local pipe name (simulator or local server)</td>
</tr>
<tr>
<td>SfCOM1</td>
<td>tty port 1 (/dev/ttya or /dev/ttyS0)</td>
</tr>
<tr>
<td>SfCOM2</td>
<td>tty port 2 (/dev/ttyb or /dev/ttyS1)</td>
</tr>
</tbody>
</table>

Table 1. Port types and Saphira-defined names for client/server connections.

The paiRobotStartup function then begins several processes for sending and receiving commands to and from the robot server. Among other tasks, the paiRobotStartup function begins the sonar ping cycle and the motor controllers. Table two shows the function call to initiate a start-up [6].

```c
PaiRobotStartup (sfTTYPORT, sfCOM1);  /* Connect through serial port COM1 */
```

Table 2. Connection example. Connecting a PAI Client with a Robot.
This example demonstrates the connection to a robot through the PAI. The disconnect
function is just as simple as shown in Table three [6].

```c
PaiRobotShutdown (void); /* Sever the connection */
```

**Table 3. Disconnecting example. Disconnecting a PAI Client from a Robot.**

Table 4 is a PAI program that connects with and then immediately disconnects from the
simulated pioneer robot [6].

```c
#include “pai.h”
void main (argv, argc)
{
    if(!sfConnectToRobot(sfLOCALPORT, sfCOMLOCAL)) /* Pioneer Simulator */
    {
        printf(“Couldn’t open simulator\n”);
        exit(0);
    }
    paiRobotShutdown (void); /* Shut it down */
}
```

**Table 4. Connect/Disconnect example. PAI connecting/disconnecting with a Robot.**

The previous examples demonstrated how the PAI is used to write programs that
make communications with the robot possible [6]. The initial goal of this project was to
discover how a CLIPS application could utilize these commands to control a robot. The
CLIPS application can simply issue a StartupRobot function call that will execute the
connect commands shown above.

Other PAI functions contain translational, rotational and status functions to allow
for the movement of the robot. The translational and rotational functions are based on a
Cartesian coordinate system measured in millimeters. C functions exist to set the
velocity, direction, and the distance to move. There is also a C function to determine the
status of the robot. The status would either be moving, stopped, or no power [6]. These
functions all exist in the PAI and are now available for use in CLIPS programs. Each of these functions formed the first priority of this project.
III. Project Tasks

3.1 Development

The steps to complete this project included learning material unknown to the developer. The understanding of several programming languages and software packages was necessary prior to development of any type of interface. Having no experience in the field of expert systems, the project developer had to educate himself on this topic. The expert-systems tool CLIPS was studied as well. The need existed during the design and testing phase to have the ability to understand CLIPS to the point of writing code. The project developer also had to have a good understanding of the Pioneer API's which is used to control the robot.

The next step was to write the interface functions so they were available to CLIPS applications. Once the functions were available, they were declared as user functions to CLIPS. An important feature of CLIPS is “the ability to integrate CLIPS with external functions or applications” [6]. In order for CLIPS programs to properly access external functions, the description of all external functions must be provided to CLIPS. “User-defined functions are described to CLIPS by modifying the function UserFunctions” [7]. For every function integrated with CLIPS, the function UserFunctions calls the DefineFunction routine. Once this is done, compilation and linking of the user's source code, of the new functions, to CLIPS can be completed [6]. Table 5 is a description of DefineFunction from the Pioneer Application Interface (PAI) Programmers Manual. Table 6 is a UserFunction declaration from the PAI Programmers Manual.
Int DefineFunction (functionName, functionType,
    FunctionPointer, actualFunctionName);

Char *functionName, functionType, *actualFunctionName;
Int (*functionPointer)();

Table 5. DefineFunction in PAI.

UserFunction()
{
    /*---------------------------------------------*/
    /* Declare your C functions if necessary. */
    /*---------------------------------------------*/
    extern double rta();
    extern void *dummy();

    /*---------------------------------------------*/
    /* Call DefineFunction to register user-defined functions */
    /*---------------------------------------------*/
    DefineFunction("rta", 'd', PTIF rta, "rta");
    DefineFunction("mul", 'l', PTIF mul, "mul");
}

Table 6. UserFunctions in PAI.

"The first argument to DefineFunction is the CLIPS function name, a string representation of the name that will be used when calling the function from within CLIPS. The second argument is the type of the value, which will be returned to CLIPS. The third argument is a pointer to the actual function. The macro identifier PTIF can be placed in front of a function name to cast it as a pointer to a function returning an integer. The fourth argument is a string representation of the third argument (the pointer to the actual C function)" [6]. The following list is the steps needed to define and implement a user-defined function according to the CLIPS Advanced Programmers Guide.

1) Copy the entire CLIPS source code file to the user directory.
2) Define the new CLIPS user function in a new file.
3) Define the constructs that use the new function in a new file (or in an existing file).
4) Modify the CLIPS main.c file or the userfunctions.c file to include the new UserFunction definition.
5) Compile the CLIPS files along with any files that contain user-defined functions.
6) Link all object code files.
7) Execute new CLIPS executable.
8) Load the construct file and test the new function.

Once the steps to define a user-defined function are implemented, the CLIPS application only needs to reference the function in order to use it [7].

3.2 Testing

Once the functions were implemented, the testing phase began. A CLIPS expert system was developed for testing purposes. The expert system was used to test the functions by having them be called from within the CLIPS program. Each function was tested individually and was also tested together.

The expert system that was developed for testing uses information returned from the robot's pingers (sonar sensors) to determine a new direction to travel, see appendix 1. This was accomplished by writing an internal function within the clips application that assigns an absolute radian direction to each pinger. Then the distance from each pinger to an object in its path is determined. The pinger with the largest distance value is the pinger used to ascertain the absolute radian direction to turn the robot. Upon completion of the testing phase, the project development was complete.
IV. CLIPS Interface Functions

The CLIPS-to-Saphira interface developed by this project is composed of six functions. The functions are declared in the UserFunctions definition file. This file must be included with any compile of CLIPS in the future. Once compiled and linked, the interface functions were available for use through the CLIPS application program. The interface functions consist of the following:

- a startup function (appendix 2),
- a disconnect function (appendix 3),
- a function that provides the ability to turn and move the robot (appendix 4),
- a function to determine the robot's current status (appendix 5), and
- a function to determine the sonar range to an object (appendix 6).

A CLIPS program could use the information provided from the sonar readings of the robot which are accessed using the SonarRangeRobot function to move the robot with the move function.

CLIPS does not actually pass the arguments to the function, it stores the arguments internally. In order to access the arguments, a call must be made to the argument access functions within CLIPS. The interface functions that were developed determine if they have received the correct number of arguments. Several CLIPS functions exist to help deal with this situation [7].

The following table summarizes the functions that were developed for use within a CLIPS application.
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client/Server Connection Functions</strong></td>
<td></td>
</tr>
</tbody>
</table>
| StartupRobot | 1. Connects the client to the robot server through the specified port.  
2. Initializes the velocity to zero.  
3. Resets position of the robot based on a Cartesian coordinate system. |
| ShutdownRobot | 1. Reset the rotational velocity to zero.  
2. Resets the translational velocity to zero.  
3. Disconnects the client from the robot server. |
| StatusRobot | Returns the status of the robot’s power and the status of the robot’s motion. |
| **Movement Functions** | |
| SonarRangeRobot | Determines the distance to the closest object. |
| StopRobot | 1. Resets the rotational velocity to zero.  
2. Resets the translational velocity to zero. |
| MoveRobot | 1. Moves the robot at a specified velocity.  
2. Turns the robot to a specified radian direction.  
3. Moves the robot a specified distance. |

Table 7. Interface functions. Functions that are available for use within a CLIPS application.

These functions utilize some of the features of the PAI that ease the connection to the Saphira environment. Some of the PAI functions were grouped together and incorporated within a single CLIPS interface function to allow CLIPS programmers to know as few commands as necessary and still have adequate control over the robot. Intelligently controlling a mobile robot with just six functions requires that the functions be versatile. The following sections describe each CLIPS interface function in detail. They are grouped by the type of function being performed.

**4.1 StartupRobot Function**

Three input parameters are required: Connection method as a string, port of connection as a string, velocity as an integer. One integer value is returned a one denoting the robot has started and a zero denoting the robot has not started. The first
parameter of the StartupRobot function allows the programmer to choose to connect the client to the robot or the simulator server.

A parameter is passed to the StartupRobot function specifying the connection method. This particular feature of the StartupRobot function is similar to the PAI function paiRobotStartup. To connect to the simulator, pass the string “sfLOCALPORT” as the first parameter to StartupRobot. To connect to the robot via TTY port, pass the string “sfTTYPORT” as the first parameter to StartupRobot. To connect to the robot via TCP, pass the string “sfTCPPORT” as the first parameter to StartupRobot.

The second parameter of the StartupRobot function allows the programmer to choose to connect the client to the robot server through the specified port or the simulator through the specified pipe. A parameter is passed to the StartupRobot function specifying the connection port. To connect using TTY port 1 or COM 1, pass the string “sfCOM1” as the second parameter to StartupRobot. To connect using TTY port 2 or COM 2, pass the string “sfCOM2” as the second parameter to StartupRobot. To connect to the simulator using a local pipe, pass pipe's full pathname as the second parameter to StartupRobot. For example, “/a1/stu/scb66618/robot_pipe” would be the path to the local pipe for the project developer's user id.

The third parameter of the StartupRobot function allows the programmer to choose the initial velocity of the robot at startup time. This is accomplished by utilizing the paiSetRobotVelocity function currently available in the PAI.

The StartupRobot function also utilizes the paiResetRobotPosition function available in the PAI to reset the robot's current position at the origin of the Cartesian plane. The
position is reset to \( x=0, y=0, \) and \( \theta=0, \) where “\( \theta \)” represents the heading measured in radians.

Upon receipt of the parameters, the `StartupRobot` function does the following:

1. Checks for the correct number of parameters.
2. Checks the parameters it receives for the correct data type.
3. Calls the `pairRobotStartup` function to start the robot/simulator.
4. Calls the `pairSetRobotVelocity` function to set the translational velocity.
5. Calls the `pairResetRobotPosition` function to reset the robot's current position to the
   origin of the Cartesian plane. This does not move the robot, it simply resets the
   coordinates to \( x=0, y=0, \) and \( \theta=0, \) where “\( \theta \)” represents the heading measured in
   radians.

An example of CLIPS code that uses this function to start the robot simulator is:

```clips
; Start the robot simulator on COM port 1
(defrule MAIN::start-robot-simulator-port1
 ?startit <- (start-robot "1")
 ?theport <- (use-port "1")

 =>
 (retract ?startit)
 (retract ?theport)
 (printout t "Using channel: sfLOCALPORT" crlf)
 (printout t "Using port name: sfCOM1" crlf)
 (printout t "Start the simulator \\
             [(StartupRobot "sfLOCALPORT" "sfCOM1" 0) crlf]
             Hit <cr> to continue \\
             )
 (printout t "
 (bind ?answer (readline)))
```

### 4.2 ShutdownRobot Function

No input parameters are required. One return value: a one denoting that it has
stopped or a zero denoting that the robot has not stopped.

The `ShutdownRobot` function possesses the following features:
a) Reset the rotational velocity to zero and reset the translational velocity to zero by utilizing the PAI function paiStopRobot.

b) Disconnect the client from the robot server via the PAI function paiRobotShutdown.

An example of CLIPS code that uses this function to shutdown the robot simulator is:

```clips
; Shutdown the robot simulator on COM port 1
(defrule MAIN::shutdown-robot-simulator-port1
  ?shutdown <- (stop-robot "1")
=>
  (retract ?shutdown)
  (printout t "Shutdown the simulator ")(ShutdownRobot) crlf
  (printout t "Hit <cr> to continue ")
  (bind ?answer (readline)))
```

### 4.3 StatusRobot Function

No input parameters are required. A value denoting power status or motion status as an integer is returned. The StatusRobot function utilizes the PAI function paiRobotStatus provides the general status of the robot.

The StatusRobot function returns to the calling program whether the robot is on or off and whether the robot is moving or not. The argument being returned is an integer data type. A four denotes power off, a two denotes that power is on and that the robot is not moving, and a three denotes that the robot is on and moving. A one was not identified.

An example of CLIPS code that uses this function to determine the status of the robot simulator is:

```clips
; Check the status of the robot
(defrule MAIN::status-robot
  ?stat <- (status-robot "check")
=>
  (retract ?stat)
  (printout t " Status of the simulator ")(StatusRobot) crlf
  (assert (status-is (StatusRobot)))
```
4.4 SonarRangeRobot Function

One input parameter is required: integer denoting which pinger to use when determining sonar range. This value is an integer in the range of zero through seven.

A value denoting the distance measured in millimeters to the closest object within range is returned as an integer data type. If there is no object within the maximum range, the maximum sonar range (about 5 meters) will be returned. The minimum sonar range is approximately 200 millimeters.

Upon receipt of the parameters, the SonarRangeRobot function does the following:

1. Checks for the correct number of parameters.
2. Checks the parameters it receives for the correct data type.
3. Calls the paiSonarRange function to determine the distance to an object.

An example of CLIPS code that uses this function to determine the distance to the object in front of pinger 6 is:

```
; Check the distance to the robot
(defrule MAIN::range-robot
    ?range <- (range-robot "check")
=>
    (retract ?range)
    (printout t " Distance from Pinger 6 is " (SonarRangeRobot 6) crlf))
```

4.5 StopRobot Function

Two input parameters are required: floating point denoting rotational velocity and an integer denoting translational velocity. There is no return code. Since the robot or simulator does not stop immediately when told to do so, the StopRobot function would have to wait for the robot to stop before determining if it actually did stop. Therefore, no return value is used.
The StopRobot function resets the rotational velocity to zero and resets the translational velocity to zero. Utilizing the paiRotateRobot and passing it a zero in a floating-point data type denoting zero radians/second will stop the rotation of the robot. Utilizing the paiSetRobotVelocity and passing it an integer zero denoting zero millimeters/second will stop the translational velocity.

Upon receipt of the parameters, the StopRobot function does the following:

1. Checks for the correct number of parameters.
2. Checks the parameters it receives for the correct data type.
3. Calls the paiRotateRobot function to stop the rotational velocity.
4. Calls the paiSetRobotVelocity function to stop the translational velocity.

An example of CLIPS code that uses this function to stop the robot simulator is:

```clips
; Stop the robot
(defrule MAIN::stop-robot
  ?stopit <- (stop-robot "stop")
  =>
    (retract ?stopit)
    (printout t " Stopping the robot " (StopRobot 0 0) crlf))
```

4.6 MoveRobot Function

Three input parameters are required: Velocity, radian measurement for direction, and distance. There is no return code for this function because having one would not be useful. Since the robot or simulator does not start to move immediately when told to do so, the MoveRobot function would have to wait for the robot to start before determining if it actually did start. Therefore, no return value is used. The MoveRobot function performs the following features:

1. Moves the robot at a specified velocity.
2. Turns the robot a specified number of absolute radians.

3. Moves the robot a specified distance.

Passing an integer denoting the velocity to the paiSetMoveVelocity will set the robot’s operating velocity. The MoveRobot function requires a call to the PAI function paiSetRobotHeading. The paiSetRobotHeading must receive from the MoveRobot function a data type float parameter containing the absolute radian measure of the direction to rotate in radians/second. The robot will then rotate until the command is reversed, the robot is stopped or the direction is obtained.

The MoveRobot function also requires a call to the paiMoveRobot function. This call must pass it an integer of a distance in millimeters, which will cause the robot to begin to move until the distance is reached.

Upon receipt of the parameters, the StopRobot function does the following:

1. Checks for the correct number of parameters.

2. Checks the parameters it receives for the correct data type.

3. Calls the paiSetMoveVelocity function to set the robots velocity.

4. Calls the paiSetRobotHeading function to set the absolute radian direction.

5. Calls the paiMoveRobot function to move the robot a specific distance

An example of CLIPS code that uses this function to move the robot simulator is:

```clips
; Move the robot
(defrule MAIN::move-robot
  ?moveit <- (move-robot "move")
=>
  (retract ?moveit)
  (printout t " Moving the robot "(MoveRobot 200 3.14 2000) crlf)))
```
V. Conclusion

5.1 Results

Upon completion of the project, the project developer was able to demonstrate the interface between a CLIPS application and a Pioneer Mobile Robot. There are several interface functions available for use through the CLIPS programming environment to control the robots. The interface functions are: STARTUP, SHUTDOWN, DISCONNECT, MOVE and STATUS.

Objectively speaking, the project developer feels the project was a success since all goals were met. The project was challenging and well developed. Some minor changes in the original design were necessary to make the project successful. These changes were in regards to which PAI function would be the best to use when moving the robot. There are many PAI functions that allow you to move the robot. The project developer had to determine which would be the best to use in the new MoveRobot function.

The new version of CLIPS that the project developer compiled allows the user of CLIPS to reference the new functions at any time. There are no extra steps a CLIPS developer needs to do in order to use these functions. The programmer simply makes a call to the function from within their CLIPS application. This makes the control of the robot using an expert system language very simple. This was the goal of the project and the project developer feels it has been very successful.

5.2 Future Work

Future goals of this project could include access to the compass, speaker, camera access, and color detection by use of the camera and determination of the angle of objects
in the path of the robot. These features should be included as functions for use by expert systems programmers just as the functions described in this report. Adding these functions would increase the control of the robot and would be an exciting opportunity for other students to accomplish.
Bibliography


Appendix 1.

StopRobot.c – Function source code.

#include "stdio.h"
#include "clips.h"
#include "string.h"

// StopRobot.c By: Sean Babcock //
// THIS PROGRAM STOPS THE ROBOT //

void *StopRobot()
{
    // THE RVELCITY VARIABLE IS THE ROTATIONAL VELOCITY
    float rvelcity;

    // THE TVELCITY VARIABLE IS THE TRANSLATIONAL VELOCITY
    int tvelcity;

    // THE IS_STARTED VARIABLE WILL DETERMINE IF EACH FUNCTION WAS SUCCESSFUL
    int isStopped;

    // CHECK THE NUMBER OF ARGUMENTS PASSED TO THIS FUNCTION
    if (ArgCountCheck("StopRobot", EXACTLY, 2) == -1) return (-1);

    // USE THE RtnDouble FUNCTION TO RETURN THE 1ST ARGUMENT PASSED AND STORE IN RVELCITY
    rvelcity = RtnDouble(1);

    // USE THE RtnDouble FUNCTION TO RETURN THE 2ND ARGUMENT PASSED AND STORE IN TVELCITY
    tvelcity = RtnDouble(2);

    // Using two functions to stop the robot instead of one (paiStopRobot) will allow
    // the user to stop either one or both at any time.

    // RESETS ROTATIONAL VELOCITY TO ZERO OR SPECIFIED VALUE
    // paiSetRobotRotVel(rvelcity);
    paiRotateRobot(rvelcity);

    // ?? CANNOT USE THIS IS_STOPPED BECAUSE IT TAKES TOO LONG FOR THE ROBOT TO ACTUALLY
    // ?? STOP BEFORE THIS PRGM CHECKS FOR THE 0 VELOCITY
    // RETURNS THE ROTATIONAL VELOCITY
    if (paiGetRobotRotVel() == 0.0)
        isStopped = TRUE;
    else
        isStopped = FALSE;

    // ?? CANNOT USE THIS IS_STOPPED BECAUSE IT TAKES TOO LONG FOR THE ROBOT TO ACTUALLY
    // ?? STOP BEFORE THIS PRGM CHECKS FOR THE 0 VELOCITY
    // RESETS TRANSLATIONAL VELOCITY TO ZERO OR SPECIFIED VALUE
    paiSetRobotVel(tvelcity);

    if (paiGetRobotTransVel() == 0.0)
        isStopped = TRUE;
    else
        isStopped = FALSE;

    return (1);
}
Appendix 2.

MoveRobot.c – Function source code.

#include "saphira.h"
#include "pai.h"
#include "stdio.h"
#include "clips.h"

// MoveRobot.c  By: Sean Babcock  //
// THIS PROGRAM MOVES THE ROBOT ACCORDING TO THE PARAMETERS SUPPLIED //

void *MoveRobot()
{
    DATA_OBJECT argument;

    //int status;
    // THE RVELCITY VARIABLE IS THE ROTATIONAL VELOCITY
    int velocity;
    // THE DIRECTION VARIABLE IS THE DIRECTION MEASURED IN RADIANS/SECOND
    float direction;
    // THE DISTANCE VARIABLE IS THE DISTANCE MEASURED IN MILLIMETERS
    // POSITIVE NUMBERS REFLECT FORWARD, WHILE NEGATIVE NUMBERS REFLECT BACKWARD
    int distance;

    // CHECK THE NUMBER OF ARGUMENTS PASSED TO THIS FUNCTION
    if (ArgCountCheck("MoveRobot", EXACTLY, 3) == -1) return (-1);

    // RETURN THE 1ST ARGUMENT PASSED - VELOCITY
    velocity = RtnDouble(1);
    // RETURN THE 2ND ARGUMENT PASSED - DIRECTION
    direction = RtnDouble(2);
    // RETURN THE 3RD ARGUMENT PASSED - DISTANCE
    distance = RtnDouble(3);

    // MOVE THE ROBOT AT A SPECIFIED VELOCITY
    paiSetMoveVelocity(velocity);
    paiMoveRobot(direction);

    // MOVE FORWARD OR BACKWARD DISTANCE THEN STOP
    // ?? THE FOLLOWING FUNCTION DID NOT WORK PROPERLY NOT SURE WHY BUT
    // ?? ALL IT DID WAS MAKE MINOR ADJUSTMENTS TO THE DIRECTION
    //paiSetRcbotHeading(direction);
    paiRotateRobot(direction);

    // ?? THE RETURN STATUS OF MOVING WILL NOT WORK BECAUSE PRGM IS TOO FAST AND CHECKS
    // ?? THE MOVING STATUS AFTER THE MOVE CMD BUT STILL SHOWS NOT MOVING
    //if ((paiRobotStatus() == 3))
    //  is_moving = TRUE;
    // else
    //  is_moving = FALSE;
    //printf("ismoving %d\n", is_moving);
    return (1);
}
Appendix 3.

StatusRobot.c – Function source code.

#include "stdio.h"
#include "clips.h"

// StatusRobot.c By: Sean Babcock
// THIS PROGRAM RETURNS THE STATUS OF THE ROBOT

int StatusRobot()
{
    // THE STATUS VARIABLE WILL DETERMINE IF EACH FUNCTION WAS SUCCESSFUL
    int status;

    // CHECK THE NUMBER OF ARGUMENTS PASSED TO THIS FUNCTION
    if (ArgCountCheck("StartupRobot", EXACTLY, 0) == -1) return (-1);

    status = paIRobotStatus();
    return (status);
}
Appendix 4.

StartupRobot.c – Function source code.

#include "pai.h"
#include "saphira.h"
#include "stdio.h"
#include "clips.h"
#include "string.h"

// StartupRobot.c By: Sean Babcock //
// This program starts the robot's motors //

void *StartupRobot()
{
    int ch_code;
    FILE *infile;
    char myplolocation[100];
    // DATA_OBJECT IS THE INTERNAL CLIPS STRUCTURE USED TO STORE ARGUMENTS
    // A DATA_OBJECT TYPE IS A TYPE DEFINITION OF THE STRUCTURE DESCRIBED ABOVE
    DATA_OBJECT temp;

    // DECLARE THE VARIABLES FOR CHANNEL AND PORT NAME AS CHAR *
    char *channel, *name;

    // THIS RETURN VALUE VARIABLE IS FOR TESTING PURPOSES ONLY AND WILL BE REMOVED
    void *returnValue;

    // THE VELOCITY VARIABLE IS THE VELOCITY THAT IS SET ON STARTUP EQUAL TO ZERO
    int velocity;

    // THE IS_STARTED VARIABLE WILL DETERMINE IF EACH FUNCTION WAS SUCCESSFUL
    int is_started;

    // CHECK THE NUMBER OF ARGUMENTS PASSED TO THIS FUNCTION
    if (ArgCountCheck("StartupRobot", EXACTLY, 3) == -1)
        // ADD TO CLIPS'S SYMBOL TABLE FOR USE (page 26 CLIPS.APQ)
        return(AddSymbol(" "));
    }

    // CHECK THE FIRST ARGUMENT PASSED FOR TYPE = STRING
    if (ArgTypeCheck("StartupRobot",1,STRING, &temp) == 0)
        // ADD TO CLIPS'S SYMBOL TABLE FOR USE (page 26 CLIPS.APQ)
        return(AddSymbol(" ");}

    // USE DTOtoString FUNCTION TO CONVERT DATA_OBJECT TO STRING AND STORE IN CHANNEL
    channel = DTOtoString(temp);

    // CHECK THE SECOND ARGUMENT PASSED FOR TYPE = STRING
    if (ArgTypeCheck("StartupRobot",2,STRING, &temp) == 0)
        // ADD TO CLIPS'S SYMBOL TABLE FOR USE (page 26 CLIPS.APQ)
        return(AddSymbol(" ");}

    // USE DTOtoString FUNCTION TO CONVERT DATA_OBJECT TO STRING AND STORE IN NAME
    name = DTOtoString(temp);

    // USE THE RtnDouble FUNCTION TO RETURN THE 3'RD ARGUMENT PASSED AND STORE IN VELOCITY
    velocity = RtnDouble(3);

    // ASSIGN THE CHANNEL TO A CODE
    if (strcmp(channel, "s/LOCALPORT") == 0) (ch_code = 1;)
    if (strcmp(channel, "s/TTPORT") == 0) (ch_code = 2;)
    if (strcmp(channel, "s/TCPFPPORT") == 0) (ch_code = 3;)
    //printfl("ch_code equals: %d\n", ch_code);
    exit;

    // DETERMINE IF SERVER OR SIMULATOR
    // CONNECT THE CLIENT TO THE ROBOT SERVER THROUGH THE SPECIFIED PORT
    // BORROWED IN PART FROM DR. DANNELLY'S WEB SITE SAMPLE PROGRAM rolib.c

25
infile = popen("echo $SAPPHIRE_COMPIPE", "+")
fscanf(infile,"%s", mypipelocation);
printf("\nMy pipelocation is: %s\n", mypipelocation);
switch(ch_code)
{
    // CONNECT TO SIMULATOR ON THE HOST MACHINE
    case 1 : pairRobotStartup(sfLOCALPORT, mypipelocation);
             pairResetRobotPosition();
             is_started = TRUE;
             break;
    // CONNECT TO ROBOT ON THE A TTY PORT
    case 2 : pairRobotStartup(sfTTYPORT, sfCOM1);
             pairResetRobotPosition();
             is_started = TRUE;
             break;
    // CONNECT TO ROBOT OVER A TCP NETWORK
    case 3 : pairRobotStartup(sfTTYPORT, sfCOM1);
             pairResetRobotPosition();
             is_started = TRUE;
             break;
    default : fprintf(stderr,"ERROR: invalid startup code\n\n");
             fprintf(stderr,"exiting program...\n\n");
             is_started = FALSE;
             exit(1);
}

// THIS NAME VARIABLE IS TEMPORARY AND WILL BE REMOVED AFTER TESTING
// TRIED REMOVING AND CLIPS WILL NOT ALLOW CONNECT -- ERRORS OUT
name = strcat(channel, name);

// THIS RETURNVALUE VARIABLE IS FOR TESTING AND WILL BE REMOVED
returnValue = AddSymbol(name);

// THIS RETURNVALUE VARIABLE IS FOR TESTING AND WILL BE REMOVED
return(returnValue);

// THIS RETURN VALUE IS USED FOR RETURNING THE FUNCTIONS EXIT STATUS
// UNCOMMENT WHEN TESTING COMPLETE
return (is_started);
Appendix 5.

ShutdownRobot.c – Function source code.

#include "stdio.h"
#include "clips.h"
#include "string.h"

// ShutdownRobot.c By: Sean Babcock //
// This program shuts the robot down and stops the motors //

int ShutdownRobot()
{
    // THE IS_STOPPED VARIABLE WILL DETERMINE IF EACH FUNCTION WAS SUCCESSFUL
    int is_stopped;

    // CHECK THE NUMBER OF ARGUMENTS PASSED TO THIS FUNCTION
    if (ArgCountCheck("StartupRobot", EXACTLY, 0) == -1) return (-1);

    // RESET THE ROTATIONAL AND TRANSLATIONAL VELOCITY TO ZERO
    if (paiStopRobot() )
        is_stopped = TRUE;
    else
        is_stopped = FALSE;

    // SEVER THE CLIENT/SERVER CONNECTION
    if ( ((paiRobotShutdown() ) && (is_stopped == TRUE))
        is_stopped = TRUE;
    else
        is_stopped = FALSE;

    return (is_stopped);
}
Appendix 6.

SonarRangeRobot.c – Function source code.

#include "stdio.h"
#include "clips.h"

// SonarRangeRobot.c By: Sean Babcock //
// THIS PROGRAM RETURNS THE SONAR RANGE TO AN OBJECT //

void *SonarRangeRobot()
{
// Returns the distance in millimeters to the closest object
// detected by the specified pinger.
// Pinger is the sonar number (0-6)
// Minimum Range about 200 millimeters
// Maximum Range about 5 meters (5000 millimeters)
// The return value is the distance to the object in millimeters or the
// maximum range if no object is detected.

// THE PINGER VARIABLE IS THE SPECIFIED PINGER TO ASCERTAIN SONAR RANGE
int pinger;

// THE DISTANCE VARIABLE IS THE DISTANCE MEASURED IN MILLIMETERS TO CLOSEST OBJECT
// IF NO OBJECT WITHIN RANGE, THE DISTANCE VARIABLE WILL CONTAIN THE MAX RANGE
int distance;

// THE IS_STARTED VARIABLE WILL DETERMINE IF EACH FUNCTION WAS SUCCESSFUL
int is_success;

// CHECK THE NUMBER OF ARGUMENTS PASSED TO THIS FUNCTION
if (ArgCountCheck("SonarRangeRobot", EXACTLY, 1) == -1) return (-1);

// USE THE RtnDouble FUNCTION TO RETURN THE 1ST ARGUMENT PASSED AND STORE IN PINGER
pinger = RtnDouble(1);

// ONLY ALLOW PINGERS 0-7
if (pinger > 7) return (-1);
if (pinger < 0) return (-1);

// GET THE RANGE FROM THE SPECIFIED PINGER
// REMOVE THIS COMMENT AFTER TESTING
distance = paSonarRange(pinger);

// THIS RETURN VALUE IS USED FOR RETURNING THE DISTANCE
return (distance);
}
Appendix 7.

ControlRobot.clp – CLIPS expert-system source code.

(defmodule MAIN(export ?all)(import ?all))

(defglobal
  ?*vel* = 100 ; this is millimeters/second
  ?*dist* = 1000 ; this is millimeters and will take two seconds if vel=25
  ?*dir* = 0 ; go straight ahead
  ?*sonarcnt* = 0 ; Count the number of times the sonars are checked
  ?*sonarcnt-max* = '10000) ; Used for limiting the robots movements
  ?*sleep-duration* = 10 ; Used for spacing out sonar reading checks
; The pingnum variable contains the pinger number with the greatest distance to an object
; now take this pinger and assign a radian direction to turn.
(deffunction new-dir (?pingnum)
  (if (= ?pingnum 0) then
    (bind ?*dir* (/ (pi) 2))
  else
    (if (= ?pingnum 1) then
      (bind ?*dir* (/ (pi) 3))
    else
      (if (= ?pingnum 2) then
        (bind ?*dir* (/ (pi) 6))
      else
        (if (= ?pingnum 3) then
          (bind ?*dir* 0)
        else
          (if (= ?pingnum 4) then
            (bind ?*dir* (/ (* 11 (pi)) 6))
          else
            (if (= ?pingnum 5) then
              (bind ?*dir* (/ (* 5 (pi)) 3))
            else
              (if (= ?pingnum 6) then
                (bind ?*dir* (/ (* 3 (pi)) 2)))
            ))))))
  (bind ?*sonarcnt* (+ ?*sonarcnt* 1))
)

(deffunction update-globals (?velocity ?distance)
  (bind ?*vel* ?velocity)
  (bind ?*dist* ?distance))

(defrule MAIN::starter
  ?init <- (initial-fact)
  =>
  (retract ?init)
  (clear)
  (bind ?count 0)
  (while (<= ?count 20)
    (printout t " " crlf)
    (bind ?count (+ ?count 1)))
  (printout t " WELCOME TO THE" crlf)
  (printout t " PAI ROBOT EXPERT SYSTEM DEMO " crlf)
  (printout t " This Pai Robot Expert System is an expert system designed to utilize the" crlf)
  (printout t " new CLIPS functions to control the mobile robot." crlf)
  (printout t " " crlf)
  (printout t " Hit <cr> to begin " )
  (bind ?answer (readline))
  (assert (screen 2)))

; Screen to capture Channel
(defrule MAIN::first-scrn
(retract ?scrn)
(printout t "Connecting to Simulator on the host machine." crlf)
(assert (start-robot "*"))
(printout t "Hit <cr> to continue " crlf)
(bind ?answer (readline))
(assert (screen 3))

; Screen to capture port name
(defrule MAIN::second-scrn
?scrn <- (screen 3)
=>
(retract ?scrn)
(printout t "Connecting to Local Pipe." crlf)
(assert (use-port "3"))
(printout t "Hit <cr> to continue " crlf)
(bind ?answer (readline)))

; Start the robot simulator on local pipe
(defrule MAIN::start-robot-simulator-pipe
(start-robot "1")
(use-port "3")
?startit <- (start-robot "*")
?theport <- (use-port "3")
=>
(retract ?startit)
(retract ?theport)
(printout t "Using channel: $LOCALPORT" crlf)
(printout t "Using port name: $COMLOCAL" crlf)
(printout t "Starting the simulator on " (StartupRobot "$LOCALPORT" "a1/stu/scb66618/robot_pipe" 0) crlf)
(assert (status-robot "check")
(assert (move-how "determine"))
(printout t "Moving the simulator " (MoveRobot 3 3 0.3) crlf)
(printout t "Stopping the simulator " (StopRobot 0 0) crlf)
(printout t "Shutting down the simulator " ShutdownRobot crlf)
(printout t "Hit <cr> to continue ")
(bind ?answer (readline)))

; Check the status of the robot
(defrule MAIN::status-robot
(status-robot "check")
?stat <- (status-robot "check")
=>
(retract ?stat)
(printout t "Status of the simulator " (StatusRobot) crlf)
(assert (status-is (StatusRobot)))

; Screen to capture movement options
(defrule MAIN::move-how
?scrn <- (move-how "determine")
=>
(retract ?scrn)
(bind ?count 0)
(while (<= ?count 25)
(printout t "crif"
(bind ?count (+ ?count 1))
(printout t "crif"
(printout t "Using pre-defined velocity, distance and initial direction, the " crlf)
(printout t "robot will use its sonars to wander around the room." crlf)
(printout t "The robot will continue to navigate the room for a duration of" crlf)
(printout t "sonar reading checks," crlf)
(printout t "sonar reading checks at an interval of" crlf)
(printout t "checks per second," crlf)
(printout t "crif")
(printout t "Currently these pre-defined values are: * crlf"
(printout t "Velocity: * ?vel* mm/sec crlf"
(printout t "Initial Distance: * ?dist* mm crlf"
(printout t "Initial Direction: * ?dir* absolute radians crlf"
(printout t "* crlf"
(assert (move-how "*"))
(printout t "Hit <cr> to continue * crlf"
(bind ?answer (readline))

; Move the robot intelligently - 1 is move-how
(defrule MAIN::move-robot-intelligent
  (or (status-is 2)
      (status-is 3))
  ?dowhat <- (move-how "*" ) ; intelligent
=>
  (retract ?dowhat)
  (bind ?count 0)
  (while (<= ?count 25)
    (printout t "* crlf"
    (bind ?count (+ ?count 1))
    (printout t "* crlf"
    (printout t "The robot will use it's sonars to travel around the room.* crlf"
    (printout t "It will travel avoiding obstacles in it's" )
    (printout t " path along the way and then stop.* crlf"
    (assert (move-robot ?vel* ?dir* ?dist*))
    (printout t "Hit <cr> to begin * crlf"
    (bind ?answer (readline))
    (assert (move-robot "yes")))

; Move the robot
(defrule MAIN::move-robot
  ?move <- (move-robot "yes")
=>
  (retract ?move)
  (printout t "* crlf"
  (printout t "Sonar Count: * ?sonarcnt* crlf"
  (printout t "* crlf"
  (printout t "* crlf"
  ; Start moving the simulator
  (printout t "Moving the simulator * (MoveRobot ?velocity ?direction ?distance) crlf"
  (printout t "Moving the simulator at a Velocity of * ?velocity crlf"
  (printout t "Direction of * ?direction crlf"
  (printout t "Distance of * ?distance crlf"
  (retract ?move-now)
  (assert (status-robot "check"))
  ; now determine which way to go by using sonars
  (assert (direction "determine"))(if (> ?sonarcnt ?sonarcnt-max"
  then
    (printout t "Sonar Count limit reached: * ?sonarcnt* crlf"
    (StopRobot 0 0)
    (retract ?move)
  else
    (assert (move-robot "yes"))))

; Determine direction by obtaining distance from each pinger
; Once this is obtained, move in the direction of the greatest value returned
; from the pingers.
(defrule MAIN::determine-direction
  (or (status-is 2)
      (status-is 3))
  ?det_dir <- (direction "determine")
=>
  (retract ?det_dir)
; tell it the pinger and return distance
; store each pingers distance to objects
(bind ?distance_0 (SonarRangeRobot 0))
(bind ?distance_1 (SonarRangeRobot 1))
(bind ?distance_2 (SonarRangeRobot 2))
(bind ?distance_3 (SonarRangeRobot 3))
(bind ?distance_4 (SonarRangeRobot 4))
(bind ?distance_5 (SonarRangeRobot 5))
(bind ?distance_6 (SonarRangeRobot 6))

(printout t "pinger 0: " ?distance_0 cri)
(printout t "pinger 1: " ?distance_1 cri)
(printout t "pinger 2: " ?distance_2 cri)
(printout t "pinger 3: " ?distance_3 cri)
(printout t "pinger 4: " ?distance_4 cri)
(printout t "pinger 5: " ?distance_5 cri)
(printout t "pinger 6: " ?distance_6 cri)

; determine which distance is bigger
(if (>= ?distance_0 ?distance_1)
  (bind ?largest_dist ?distance_0)
  (bind ?pingnum 0)
else
  (if (>= ?distance_1 ?distance_0)
    (bind ?largest_dist ?distance_1)
    (bind ?pingnum 1)))

(if (>= ?distance_2 ?largest_dist)
  (bind ?largest_dist ?distance_2 )
  (bind ?pingnum 2))

(if (>= ?distance_3 ?largest_dist)
  (bind ?largest_dist ?distance_3 )
  (bind ?pingnum 3))

(if (>= ?distance_4 ?largest_dist)
  (bind ?largest_dist ?distance_4 )
  (bind ?pingnum 4))

(if (>= ?distance_5 ?largest_dist)
  (bind ?largest_dist ?distance_5 )
  (bind ?pingnum 5))

(if (>= ?distance_6 ?largest_dist)
  (bind ?largest_dist ?distance_6 )
  (bind ?pingnum 6))

: GO STRAIGHT IF THE LARGEST DISTANCE IS THE SAME AS PINGER 3'S DISTANCE
(if (= ?distance_3 ?largest_dist)
  (bind ?largest_dist ?distance_3 )
  (bind ?pingnum 3))

(printout t "Go towards pinger: " ?pingnum cri)
(printout t "Direction is: " ?dir cri)
(new-dir ?pingnum)
(printout t "Direction is now: " ?dir cri)
(assert (move-robot ?vel ?dir ?dist)))

(system clear)
(reset)
(run)