MAE 576 Mechatronics

Final Project: Distributed Sensing and Control for Mobile Robot: Automated Guided Vehicle

Group D:

John Eddy

Gaurav Tyagi

Gaurav Kimothi

Amol Kulkarni

8th May 2003

Table of Contents:

Abstract

This report discusses the implementation of a distributed sensing and control framework for a mobile robot. The project explores the advantages of reducing computational load on individual processors. The control framework consists of three processors which are coupled together with wired and wireless communication channels. We interfaced infrared (IR), radio frequency (RF), and serial communication channels between the processors. This report discusses the challenges faced while interfacing three processors. The report includes the PBasic and MATLAB source codes used for interfacing as part of the Appendix.

Introduction

The main objective of the project was to use a distributed control framework where two or more processors share the computational load. This sharing of load necessitates the need of some form of communication protocol between the processors. We developed the required framework keeping in mind a specific application. The application we chose was an Automated Guided Vehicle (AGV). The main aim of the project was to control the movement of an AGV from a Base station. There exists two way communications between the base station and AGV. The base station directs the path of the AGV and the AGV sends back the information to the base station regarding its current status. The communication that exists between these two processors is wireless. In order to overcome the computational shortcomings of the Basic Stamp we interfaced a serial communication protocol between the processor on a personal computer and the Basic Stamp. Some of the important features of the project were

- Setting up the IR communication between the two Basic Stamp processors using the Firestick II;
- Setting up the RF communication between the two Basic Stamp processors using TWS-434 433MHz RF transmitter and RWS-434 433MHz RF receiver;
- Setting up Serial communication between the Basic Stamp Processor and the processor on the Personal Computer;
- Building the Robot using the Boe-Bot Kit; and
- Calibrating the servomotors for accurate displacement and turning.

The following section discuses the hardware and software used in this project.

Hardware

- Basic Stamp II: A BASIC Stamp is a single-board computer that runs the Parallax PBASIC language interpreter in its microcontroller. The developer's code is stored in an EEPROM, which can also be used for data storage.
- Basic Stamp II processor on Board of Education (BOE).
- Various resistors
- RS -232 communication cable
- Firestick II IR Transmitter and Receiver
- 2 RF Transmitters, Receivers and Antennas
- Personal computer
- 9V Battery for IR communication transmitter
- Adapter to supply power to Basic Stamp II (12 VDC, 1000mA)
- 4 , AAA batteries to supply power to the stamp processor and servo motors on the Boe-Bot

Software

- Basic Stamp Editor
- Circuit Maker
- MATLAB

Application Setup

We used the Boe-Bot (robot built out of the Parallax robot kit) as our automated guided vehicle and the Basic Stamp II board as the base station for our application. The Boe-Bot has a Basic Stamp processor installed on it. We had 8 stations defined on a shop-floor. The stations were laid in the form of an octagon as shown in Figure 1. The center of the octagon is the origin of the world coordinate system. All the station coordinates are defined with respect to this origin. The objective was to be able to direct our mobile robot to any of these stations using the base station. For this purpose we needed to set up the 2-way communication between the two processors. Initially we had IR and RF wireless type communication setup between the AGV and the Base Station. However, we were experiencing interrupted communication through IR channel because we were not

always able to maintain line-of-sight between the emitter and the detector. Hence, we installed RF communication channels both ways.

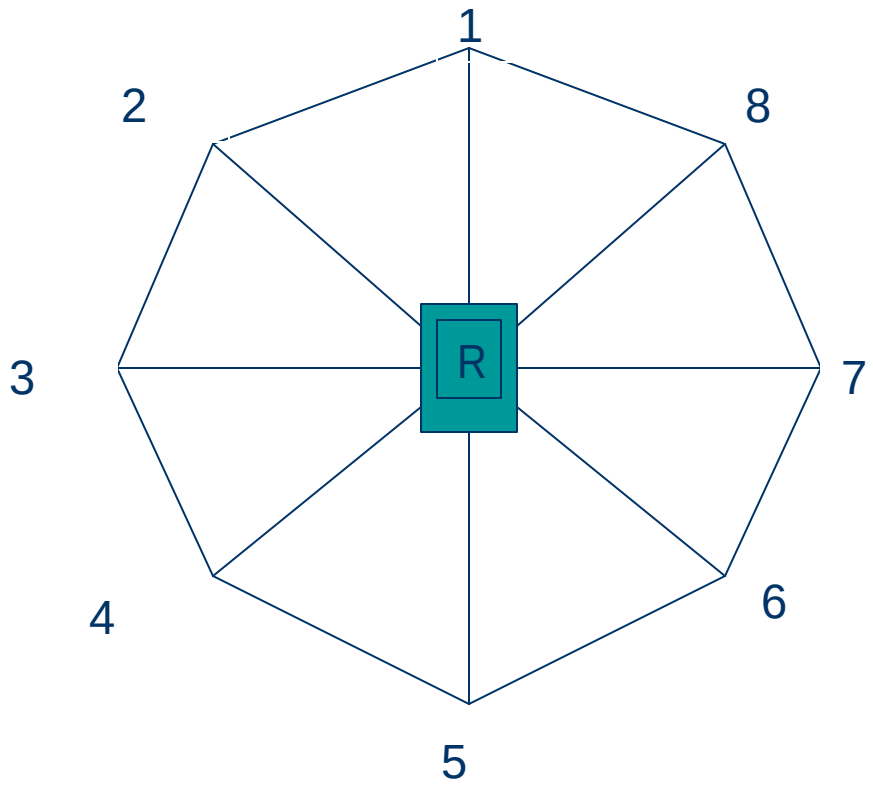

Figure 1: Octagonal Layout of AGV Stations.

Modes of Operation

There are three ways in which we can control the movement of our AGV using the Base Station.

- *Real-Time Control*: Here the Base station directs the AGV to move to a specific station. After the move completion, AGV sends back the confirmation of the move and awaits further instructions.
- *Pre-Defined Path*: Here the user can define a set path for the AGV to traverse. The AGV traverses the path without accepting any instruction in between and reports the confirmation of each move completed.

- *User Defined Coordinates*: Here the user specifies the coordinates of a point relative to the WCS origin and the AGV traverses to that point and sends back conformation of move completion and awaits further instructions.

The concept used to drive the robot is very simple. We used vector algebra to compute the current direction of the robot and the angle and distance it would need to travel in order to reach a desired station from its current position. Due to the shortcomings of the Basic Stamp processor we could not do most of the computation required to drive our robot. For example, we could not calculate the angle between the current vector and the next desired vector. This is because the Basic Stamp does not handle floating point and signed numbers well. In order to overcome this we interfaced the processor on a personal computer with the Base Station processor. We used MATLAB to do most of our computation and used the serial communication cable RS-232 to set up the serial communication between the Base station and MATLAB. The communication flow between MATLAB, the Base Station, and the Robot is shown in Figure 2(a).

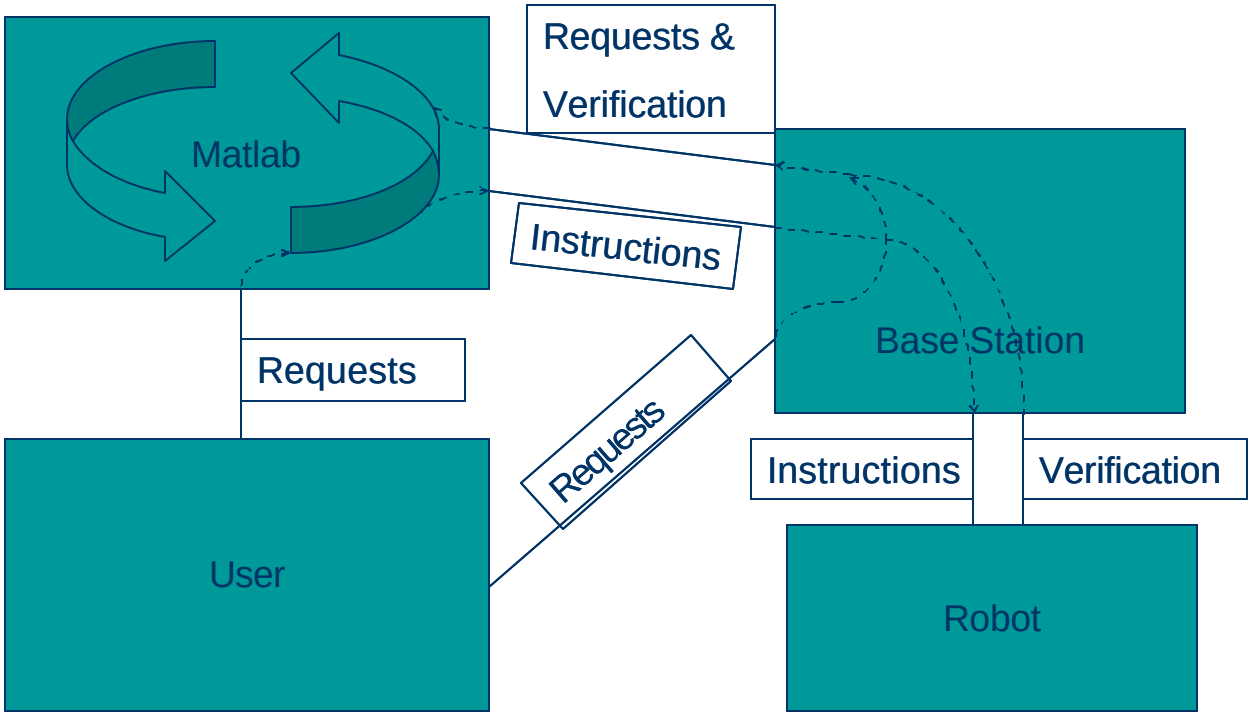

Figure 2(a): Communication Diagram for Agents in Sensing Network

The concept used to move the robot from one station to the other is described below in steps with help of Figure 3.

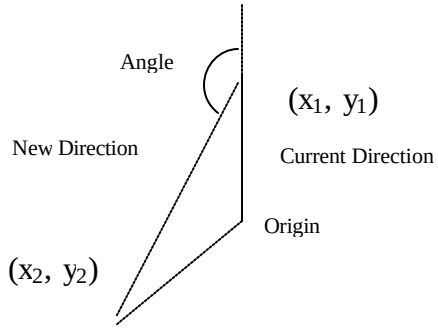

Figure 2 (b)

- Figure 2(b) shows the vector representation of the current and new or desired direction of the robot.
- Knowing the co-ordinates of the present and desired location we calculate the angle between the vectors using the dot product.
- The angle thus obtained is the acute angle between the two vectors and hence we need to decide which way to turn, for the case shown in figure the answer is left.
- In order to compute the correct direction we first calculate the unit vector along the current direction and then rotate it clockwise (right) and check if the unit vector obtained is same as that along the new direction. If it is, then the direction to turn is right otherwise left.
- We compute the distance robot needs to travel using the distance equation knowing the coordinates of the current and desired position.
- Once we have the angle, direction and the distance robot needs to travel we compute the angle cycles, distance cycles we need to send out to robot using our calibration equation explained further.

Laboratory Procedures

In this section, we discuss the various tasks that were completed on behalf of this project.

Building the Boe-Bot

The assembly of the Boe-Bot was done by following the instructions provided in the "*Robotics! Student Workbook"*. The assembly started off with the setting up of the top-side hardware on the Boe-Bot chassis. Then the two parallax pre-modified servos were mounted on to the Boe-Bot chassis after their horns had been removed. A 4-battery pack was also attached to the chassis. The Boe-Bot wheel parts were assembled and the

wheels attached. The two front wheels were attached to the servo output shafts. The supplied plastic ball was used as the tail wheel with the cotter pin as its axle. Finally the Parallax Board of Education was mounted on top of the chassis using ¼ inch screws. The servos were plugged into the servo ports on the board of education and the Boe-Bot had been completely assembled.

Setting-Up the Infra-Red (IR) Communication

Asynchronous Serial IR Communication between the Base-Station and remote-robot was implemented using the Firestick II. The transmitter was set up on the Boe-Bot while the Base-Station had the receiver set up. The figure below shows the circuit set up for IR communication.

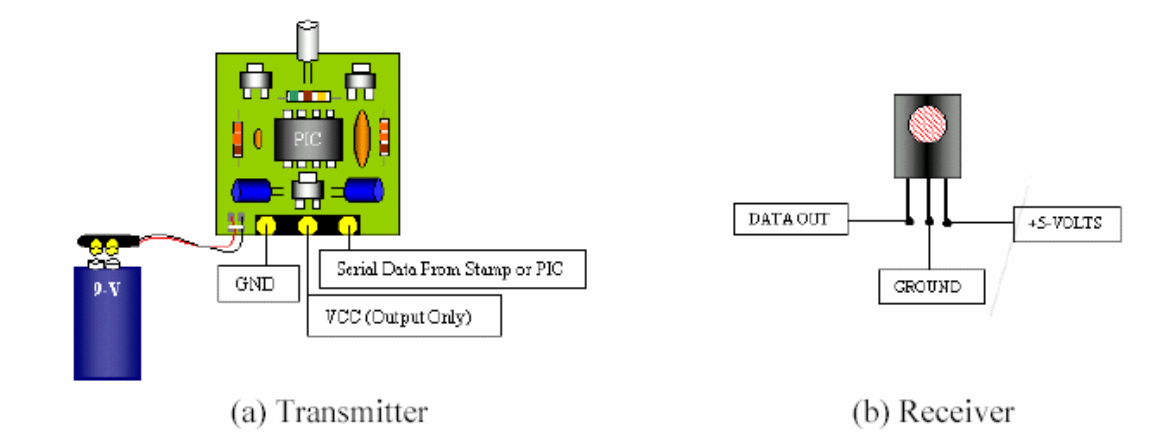

Figure 3: IR Communications Components and Setup.

The Firestick II requires a 9V battery source. It transmits serial data by connecting to a single I/O data pin on a microcontroller. The receiver circuit needs just an infrared detector module, the Panasonic PNA4602M, which is capable of serial data reception at baud rates up to 2400. The circuit connections are pretty simple; they are just connected to I/O pins and voltage sources. We chose this communication interface because both the hardware and software interface is easy.

SERIN and SEROUT commands are used to transfer data serially. The syntax for sending and receiving data is as follows:

<u>Sending data:</u>

SEROUT pin number, baudmode,["password","sent data"] (1)

Receiving data:

SERIN pin number,baudmode,[WAIT("password"),received data] (2)

By using a synchronization or address technique, the receiving controller can be forced to ignore data not preceded by a unique address or synch byte. This allows *selective control* of multiple receivers from a single transmitting station.

Setting-Up the Radio-Frequency (RF) Communication

An effective RF communications network was set up using the BASIC Stamp with the TWS/RWS RF modules. Two 433MHz whip style antennas are also used in the set up for long range detection. The receiver in this case was set up on the Boe-Bot and the transmitter was set up on the base station. The figure below shows the circuit set up.

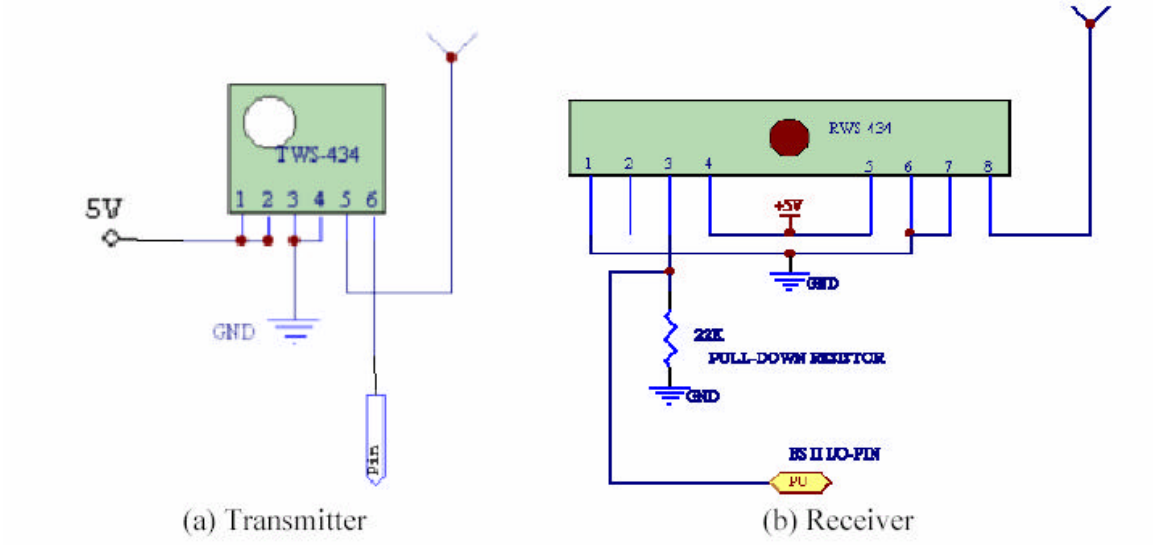

Figure 4: RF Communications Components and Setup.

The TW-434 outputs up to 8mW at 433.92 MHz. It has an operating range of about 400 ft. outdoors, or about 200 ft. indoors. It can go through most walls. The operational voltage varies from 1.5 to 12 V and it accepts both linear and digital input. Figure 5 below shows the schematic of the transmitter with its pin specifications

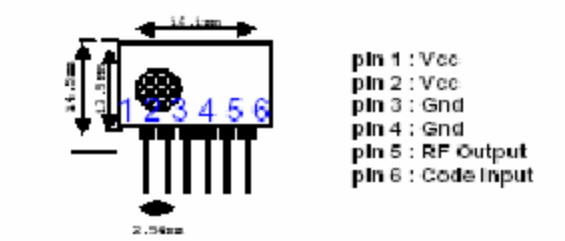

Figure 5: RF Transmitter Schematic

The RWS-434 receiver also operates at 433.92 MHz with an operational voltage of around 4.5 – 5.5 V DC. Its sensitivity is 3 µV, and it can have both linear and digital outputs. Figure 6 below shows the schematic of this receiver with the pin specifications

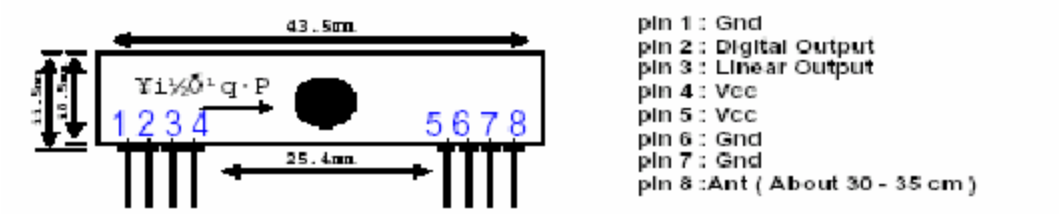

Figure 6: RF Receiver Schematic

The RF transmitter and receiver need to be connected to antennas when they are used for long-range detection. We used two 433MHz whip style antennas. This antenna is fed with an RG-174 coaxial cable. Figure 7 below shows its schematic and dimension details

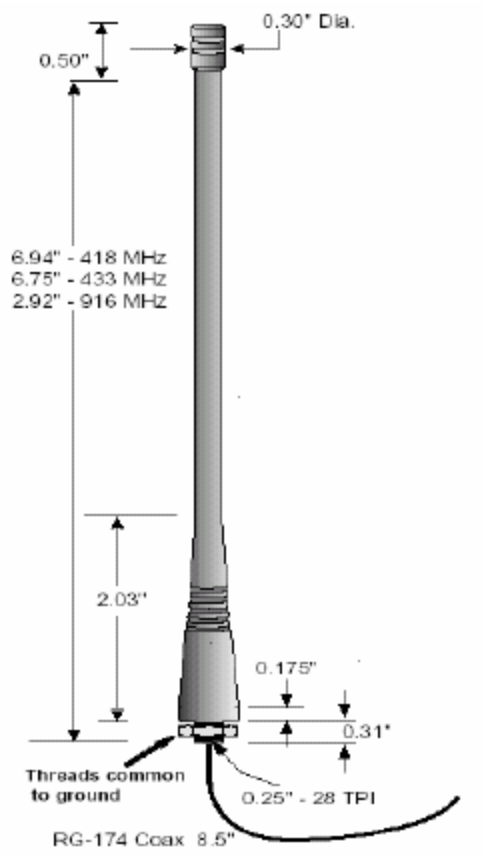

Figure 7: RF Antennae Schematic

Normally in the case of more than one set of RF communication set-ups, disturbance and interference from other RF communications may occur. To avoid this, an encoder and decoder can be used. We have as part of our kit the *Holtek 8-bit encoder HT-640 & decoder HT-648L*. The figure below shows their schematics

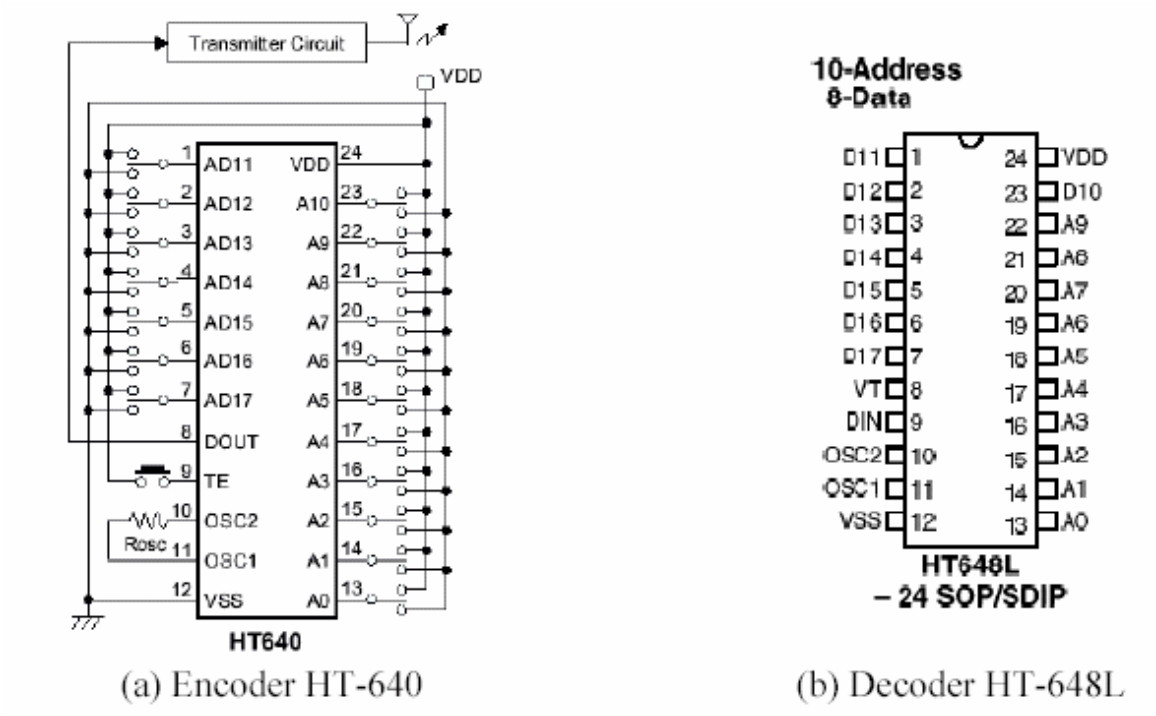

Figure 8: Encoder and Decoder Schematics

It is called the 318 Series because it sends 10 bits of address and 8 bits of data three times. This encoder and decoder pair can be assigned ten-bit addresses, so that only this pair of devices can communicate with each other. The encoder can send serial data three times after the address was sent. The serial data can also contain a qualifier (password) to increase security. Correspondingly, the decoder will have to receive the same address, same serial data three times, and the same qualifier in the specific baud mode. Therefore, even though other RF devices are operating on the same frequency, no interference occurs. Also the encoder and decoder pair operate under 2.4 V – 12 V at low power and high noise immunity. It has low standby current and is a small electronic component. However, in this project we did not use this method project. We used the 'wait' function of the SERIN and SEROUT command to make sure that the communication is not interfered. We also made use of PAUSE statements at the appropriate places in the code to account for uninterrupted data communication.

Calibrating Servos

It was very essential to calibrate the servos in order to ensure accurate movement of the robot. The robotics manual was used as a reference while calibrating the servos. The servos were attached to pins 12 and 13 of the Boe-Bot board. The Basic Stamp sends out pulses to the servo motor through pins 12 and 13 which enable the servos to rotate. High time is the main ingredient for controlling a servo's motion, and it is most commonly

referred to as the pulse width. According to the robotics manual, pulse widths for pre-modified servos range between 1.0 and 2.0 ms for full speed clockwise and counterclockwise respectively. We use the standard PBasic command PULSOUT to send out pulses to the servo. The PULSOUT value of 500 corresponds to 1.0 ms and PULSOUT value of 1000 corresponds to 2.0 ms. The pulses are sent every 20 ms in order to ensure that the servo rotates continuously. During the process of calibration, it is important was to determine the threshold value above which the servo changes its direction of rotation. After performing some experiments and using trial and error we determined that at a PULSOUT value of 750, our servos stand still and anything below or above this value makes them rotate though in opposite directions. Later we calibrated our robot to calculate the number of cycles required to make it traverse a certain distance and turn through a certain angle. We observed that among other things, the calibration was highly dependent on the power source. Hence it is imperative that the robot is calibrated periodically and with a new set of batteries. We calibrated the robot numerous times as the batteries kept dying; one such calibration chart is shown in Figure 9 below.

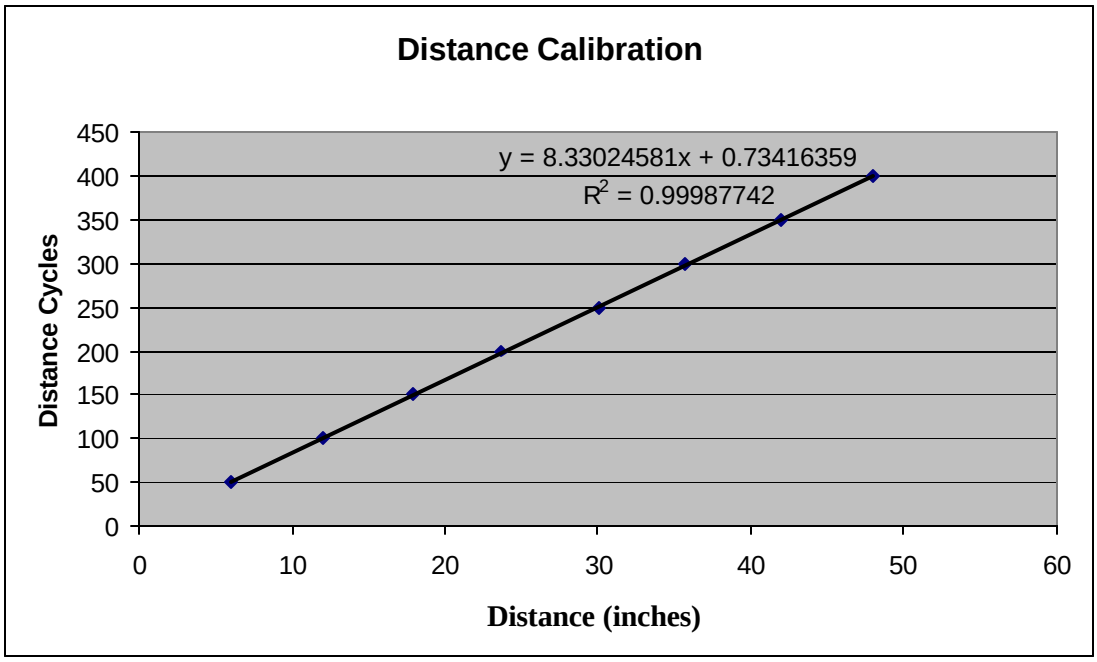

Figure 9: Example Servo Calibration Curve for Distance Cycles.

The chart in Figure 10 shows the calibration of the robot for left and right turns. The equation on the left shows the calibration equation for the left turn and the regression equation on the right shows the calibration for a right turn. Though we usually calibrated in this way, ultimately we decided to use a single equation for both turn directions.

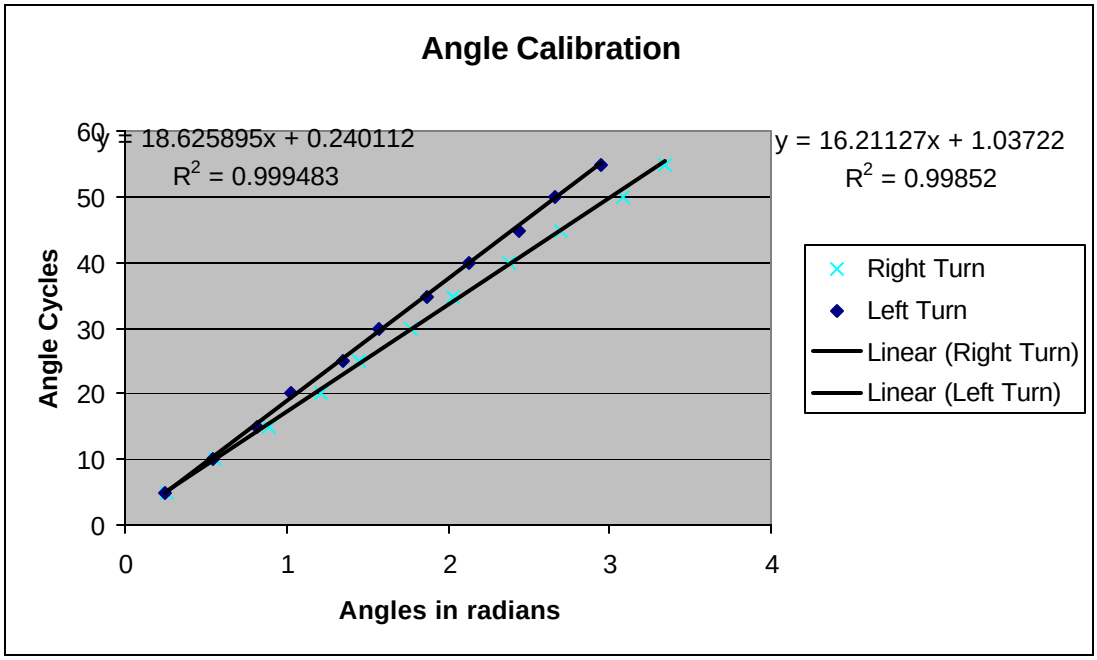

Figure 10: Example Servo Calibration Curve for Angle Cycles.

Programming

A unique program was written for each of our three processors. A listing of the code can be found in the appendices of this document. The code given to the Boe-Bot was designed to perform the requested movements and report status such as "message received" and "move completed".

The program for the base station was intended to facilitate communication between MATLAB, the user, and the Boe-Bot. So in that respect, it is little more than a relay station.

Finally, the MATLAB code is primarily used to compute the required moving instructions to achieve a users request of the Boe-Bot. These instructions, once computed, are sent through the Base-Station and onto the Boe-Bot.

Figures 11 through 13 below show an overview of the flow of control of each program. Specific functions are not represented. The diagrams are intended only to give a basic idea of the operations of each unit.

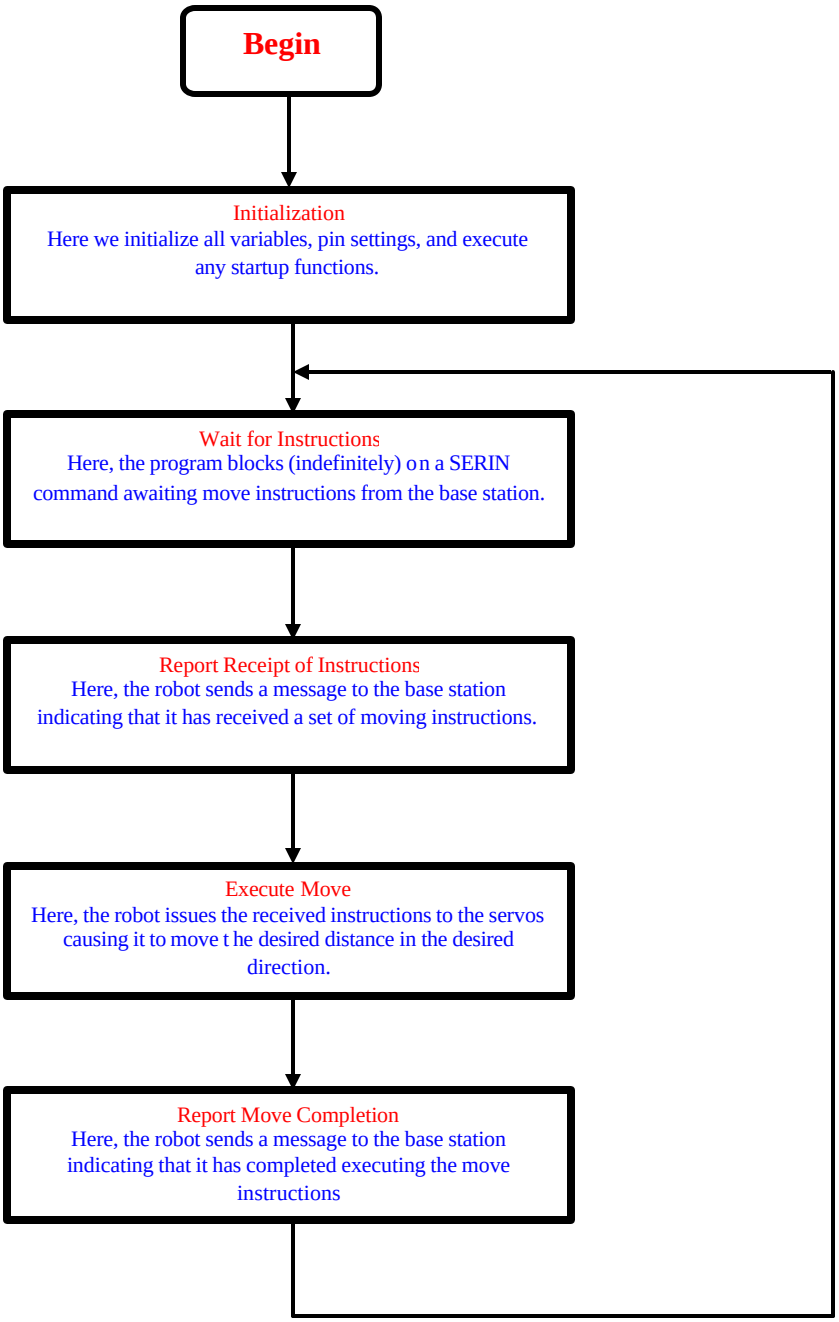

Figure 11: Robot Source Code Flow of Control.

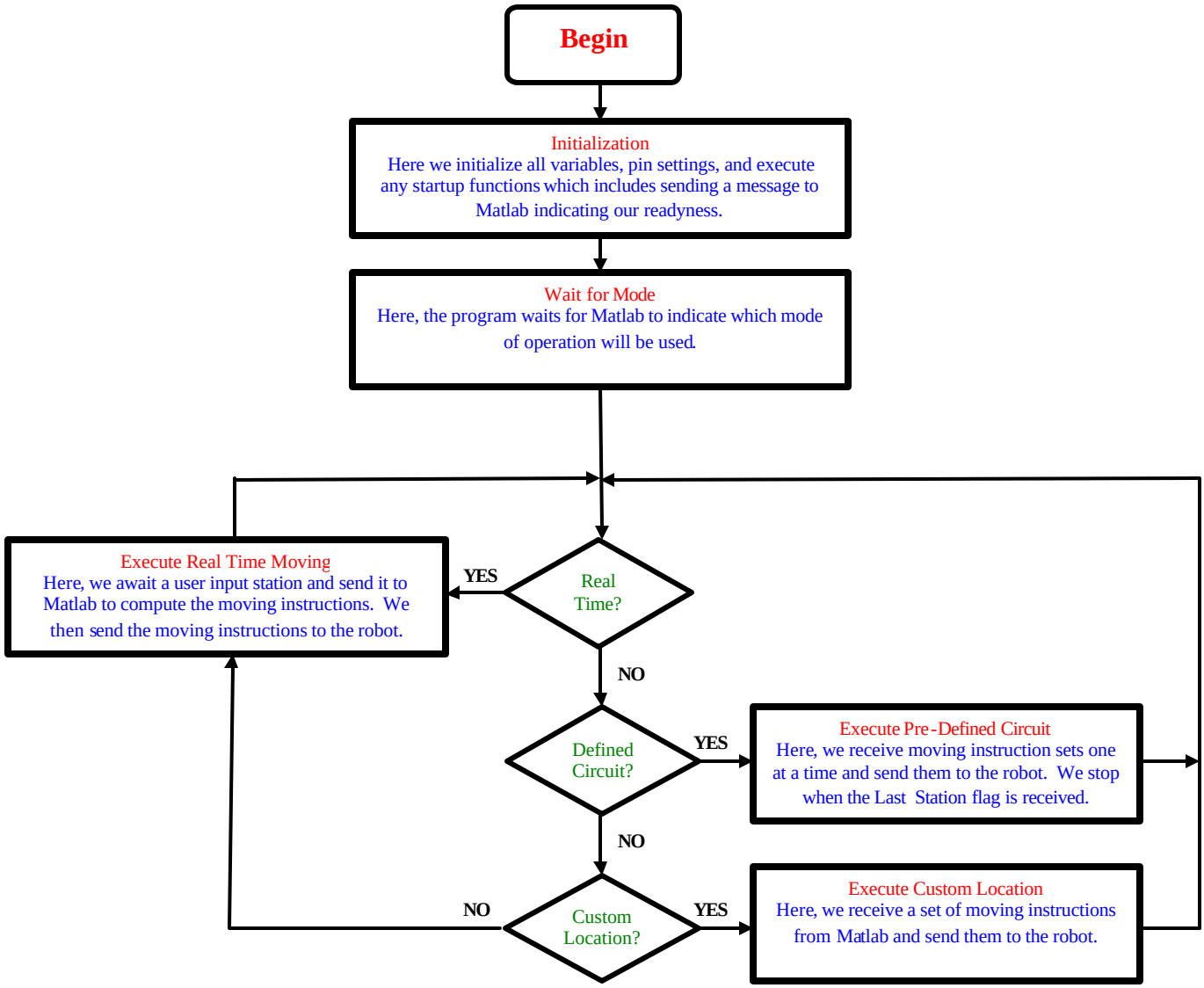

Figure 12: Base Station Source Code Flow of Control.

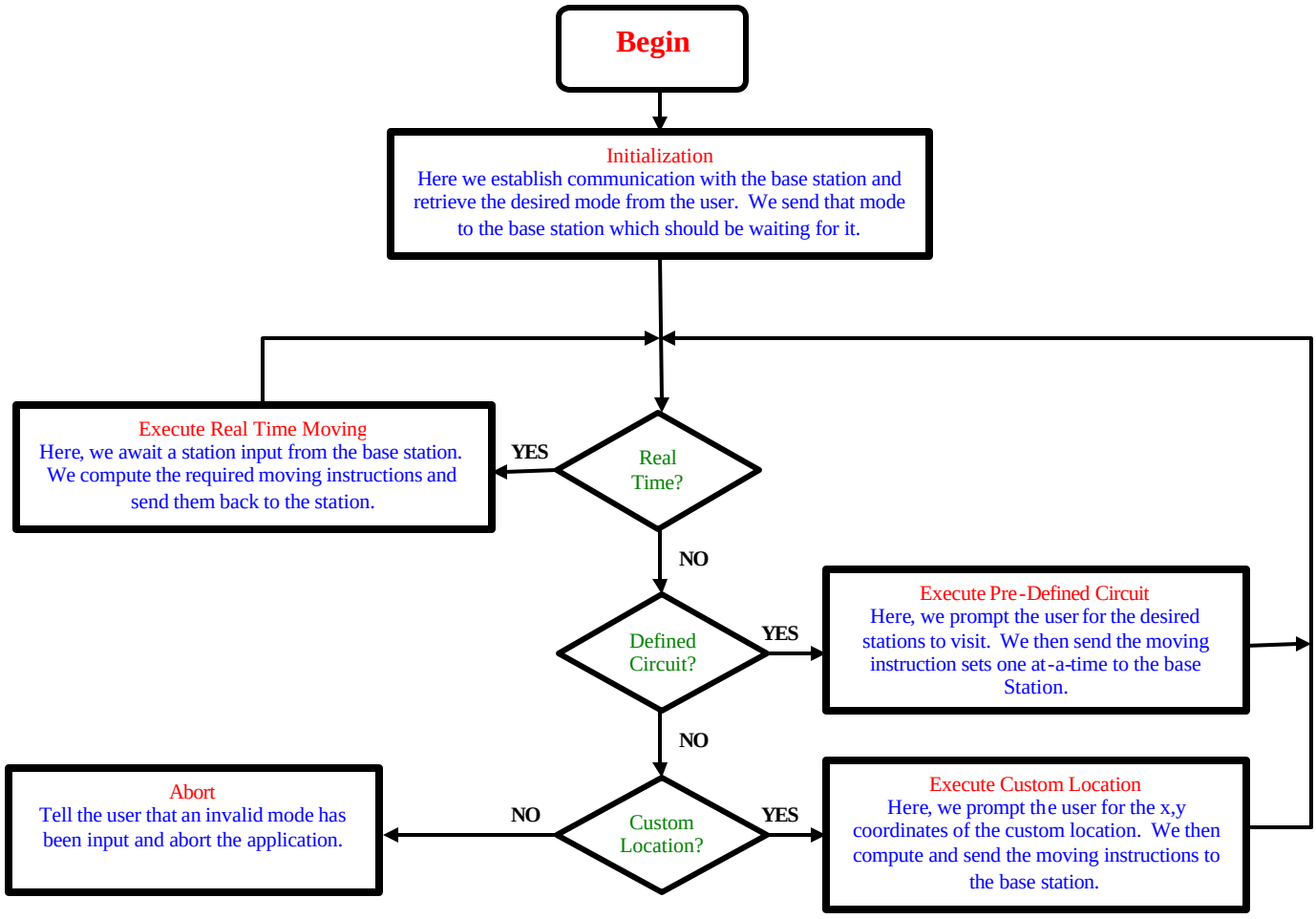

Figure 13: MATLAB Source Code Flow of Control.

Results

We built an AGV which is controlled from a Base-Station using two way wireless RF communications. We were able to control the movement of our AGV from the Base-Station in three ways.

- Real-time control using the Base-Station push buttons as input from the user.
- Pre-defined path for the AGV to traverse using MATLAB to acquire input.
- User defined specific coordinates of a point relative to the WCS origin. The AGV traverses to that point.

The AGV is capable of achieving accurate travel to a station within a margin of about 3-6 inches over a 48 inch distance. Some observed sources of error are:

- Battery Charge;
- Irregularities on the floor such as differences in friction and elevation;
- Slight differences in time between the activation of the two servos; and
- Inertia of the Boe-Bot (affecting primarily the angle-of-turn accuracy).

Discussion

This section of the report discusses the difficulties we faced in the project and the steps we took to overcome them. It also discusses the level of success achieved in the project. The initial setup of the project where we had to assemble the robot and set up the various communication channels was very smooth. We could transmit and receive data using the channels fairly accurately. Initially we had the IR transmitter mounted on the base station and the IR receiver on the robot. We realized that since at times the robot leaves the line-of-sight of the base station, we encountered interrupted communication between the processors. In which case the base station would leave communication with the robot and will not be able to control it. So we thought of interchanging the communication setup and had the IR transmitter mounted on the robot and RF transmitter on the Base-Station. In this case even if our robot is unable to communicate with the Base-Station, the Base-Station could easily send instructions to the robot using the RF communication and hence if for a while the Base-Station receives no feedback from the robot, the Base-Station could ask the robot to come back to the Base-Station. But during our laboratory procedures we encountered too many problems having to communicate using the IR communication channel and decided to replace it with another RF communication set.

In our project we were required to keep the information of the current direction of the robot at all times and calculate the new direction using vector algebra and trigonometric functions. So the next difficulty we encountered was that of performing such computations using the Basic Stamp. We tried working our way through the limitations and were finally able to get the angle between the current direction vector and the new direction vector but since the angle we obtain between the vectors is always acute we had to decide which way to turn. We were unable to implement our logic and algorithm for computing the direction on the Basic Stamp. We finally searched through the net and found that we could interface the Basic Stamp with MATLAB through the serial COM port on the back of a computer. Interfacing MATLAB with the Basic Stamp also came with its

share of difficulties. It took us considerable amount of time to figure out that we were sending data in the ASCII format using the 'fprintf' command. Finally we figured out the way to send data in binary format using the 'fwrite' command and interfaced MATLAB with the Basic Stamp to handle most of our computation.

Another important feature of our project was having accurate calibration for our servo motors. We needed the robot to turn the exact number of degrees and travel exact amount of distance as computed through our algorithm. We realized that our calibration was highly dependent on the strength of the batteries we were using and hence we needed to calibrate our robot every time before testing. Therefore it was important to calibrate the servos with new batteries and always have the batteries replaced as they start to die.

In the end, we were very successful at achieving our goal of controlling the AGV. We were able to implement each of the desired modes of control and were able to calibrate the robot such that it could travel to a destination location accurately. On rare occasion, our communication paradigm fails and the result is that the robot spins almost indefinitely (it would stop eventually if we let it go). What was happening was that the timing between sends and receives of data was not perfect. So the Base-Station would begin receiving data after it had been partially sent. This happened fairly frequently at first but after properly inserting pauses, we were able to reduce the occurrence of this problem to approximately once every 20 or so times we ran the program.

Conclusions:

We were able to successfully coordinate the communication between the three processors and as a result, were able to successfully navigate the Boe-Bot. Though we had some trouble with IR communication, calibration, and computation on the Basic Stamp, the project went smoothly and no insurmountable difficulties were encountered.

Contributions:

Each member had a hand in each aspect of this project. Some of the code was taken directly from the previous lab and Stamp Manual experiments. Each member was concerned with properly interfacing the three processors and each member participated in determining solutions to the difficulties which we encountered.

Appendix I: Matlab Source Code

```
% This is the main entry point for the application.
% It prompts the user for the mode of operation and then calls the
% appropriate functions to get the process going.


% open the connection with the serial port (RS-232)
[sp, Error] = EstablishComm('COM1', 4800, 120);

if(Error == 0)
       disp(' ');
       disp('Enter the mode of operation from the list below:');
       disp('    1: Real-time station selection');
       disp('    2: Pre-programmed circuit');
       disp('    3: User input destination');
       OpMode = input(': ');

    if((OpMode > 3) | (OpMode < 1))
        disp('Invalid mode of operation.');
    else

        % These are the initial direction and location of the robot.
              NewLoc = [0, 0];
              NewDir = [-1, 0];

        % this communication tells the stampworks kit which mode has been chosen
        bs_fwrite(sp, OpMode, 'uint16', 'async');

        % execute this loop for as long as no errors occur.
              while (Error == 0)

            switch OpMode
             case 1,
                    % Use buttons on stampworks to choose stations
                    Station = WaitForStation(sp);
                    [NewLoc, NewDir, Error] = TraverseStations(sp,Station, NewLoc, NewDir);
             case 2,
                    % Allow user to input stations which will then be traversed in order
                    Stations = RetrieveStations;
                    [NewLoc, NewDir, Error] = TraverseStations(sp,Stations, NewLoc, NewDir);
             case 3,
                    % Allow the user to input a destination x,y location and go to it
                    [x, y] = RetrieveLocation;
                    [NewLoc, NewDir, Error] = GoToLocation(sp,x, y, NewLoc, NewDir,1);
            end

               end
    end
end
% close the connection with the serial port (RS-232)
fclose(sp);
```

```
function bs_fwrite(com, value, prec, mode)

% This function handles the actual writing of bits to the RS-232 serial port.
% It is required that values be small enough for 16 bit representation.  It sends
% the data out one byte at a time and checks for a verificaion of transmission reply.


% Check for 16 bit overrun
if(value > 65535)
    error('Value too large in bs_fwrite.  65,535 max');
end

% Write out the first 8 bits and wait for the port to free up
fwrite(com, value, prec, mode)
while(strcmp(get(com, 'TransferStatus'), 'idle') ~= 1)
end

% Write out the second 8 bits and wait for the port to free up
fwrite(com, bitshift(value, -8), prec, mode)
while(strcmp(get(com, 'TransferStatus'), 'idle') ~= 1)
end

% get verification
fscanf(com, '%i');
```

```
function Angle = ComputeAngle(currd, nextd)

% This function computes the angle in radians between the two input vectors.
% It requires that the vectors be of length 2 meaning x,y.

if(length(currd) ~= 2)
    error('Current Direction Must Contain 2 Entries');
end
if(length(nextd) ~= 2)
    error('Next Direction Must Contain 2 Entries');
end

Angle = acos((nextd*currd') / (norm(currd)*norm(nextd)));
```

```
function AnglCycles = ComputeAngleCycles(Angle, TurnDir)

% This function converts the Angle in radians into the angle cycles required by the robot servos.
[m, b] = GetAngleCycleCoeffs;

AnglCycles = max([m*Angle - b, 0]);
```

```
function TurnDir = ComputeDirection(CurrentDir, NextDir, Angle)

% This function determines which direction to turn.  That information cannot be determine while
% calculating the Angle.  It is determined here by comparing a unit vector in the current direction
% rotated positively by "Angle" about the z-axis to a unit vector in the known new direction.  The
% rotation is computed using the standard rotation transformation matrix.

% Positive rotation amounts to a left hand turn and negative to a right hand turn.

TurnDir = 0;

% Verify the inputs
if(length(CurrentDir) ~= 2)
    error('Current Direction Must Contain 2 Entries');
end
if(length(NextDir) ~= 2)
    error('Next Direction Must Contain 2 Entries');
end
if(length(Angle) ~= 1)
    error('Angle Must be a Scalar');
end

% Calculate the unit vector in the current and next direction.
curruv = CurrentDir./norm(CurrentDir);
nextuv = NextDir./norm(NextDir);

% find the rotation of +Angle about the z-axis
temppt = ([cos(Angle), -sin(Angle); sin(Angle), cos(Angle)] * curruv')';

% if the uv's match up, we have positive rotation and we turn left. Otherwise right.
if(ourround(temppt, 12) == ourround(nextuv, 12))
    TurnDir = 1;                                    % Turn Left
else
    TurnDir = 0;                                    % Turn Right
end
```

```
function DistCycles = ComputeDistanceCycles(Inches)

% This function computes the Distance in inches to cycles required by the robot servos.

[m, b] = GetDistanceCycleCoeffs;
DistCycles = max([m*Inches + b, 0]);
```

```
function [AnglCycles, DistCycles, TurnDir, NewDir] = DirectionCalc(NewLoc, CurrentLoc, CurrentDir);

% This function encapsulates the computation of the new direction data.  Some interim data is
% used but the data of primary interest is returned.

% figure out the next_*'s for the new station from the current station.
NewDir = NewLoc-CurrentLoc;

% Get the angle between the current and next directions
Angle = ComputeAngle(CurrentDir, NewDir);

% Figure out which way to turn, left or right
TurnDir = ComputeDirection(CurrentDir, NewDir, Angle);

% Convert the angle from radians into motor cycles.
AnglCycles = round(ComputeAngleCycles(Angle, TurnDir));

% Compute the distance that must be traveled
Distance = norm(NewLoc - CurrentLoc);

% Convert the distance from inches into motor cycles
DistCycles = round(ComputeDistanceCycles(Distance));
```

```
function [comm, Error] = EstablishComm(port, baud, timeout)

% This function establishes communication with the base station through the serial port
% designated by port.

Error = 0;

% Open the serial port for communication with the base station
comm = serial(port, 'BaudRate', baud);
fopen(comm);

% allow timeout seconds between inputs
set(comm, 'Timeout', timeout);

Continue = 1;

while(Continue == 1)

    % Alert the user that we are trying to establish communicaiton.
    disp('Attempting to establish communication with the Basic Stamp...');

        % Verify communicaiton with the base station
        [A,COUNT,MSG] = fscanf(comm, '%i');

    % if fscanf failed, the reason will be indicated within MSG
        if(length(MSG) == 0)
         disp('Communication established, receiving data...');
         Continue = 0;
         Error = 0;
        else
         disp('Communication was not established.  The following message was returned:');
         disp(' ');
         disp(MSG);
         disp(' ');

         disp('How would you like to proceed:');
         disp('    1: Try again');
                disp('    2: Abort');
                ToDo = input(': ');

         switch ToDo
             case 1,
                 disp('Trying Again...');
                 Continue = 1;
             case 2,
                 Error = 1;
                 Continue = 0;
             otherwise
                 Error = 1;
                 Continue = 0;
         end
        end
end
```

```
function Error = ExecuteMove(Port, AnglCycles, DistCycles, TurnDir, LastStation)

% This function executes the provided move instructions by sending them to the base
% station to be passed on to the bot.

% Send the moving instructions back to the base station.
bs_fwrite(Port, LastStation, 'uint16', 'async');
OurPause(0.1);

bs_fwrite(Port, TurnDir, 'uint16', 'async');
OurPause(0.1);

bs_fwrite(Port, AnglCycles, 'uint16', 'async');
OurPause(0.1);

bs_fwrite(Port, DistCycles, 'uint16', 'async');

% Wait to recieve word that robot has reached its destination.
Error = WaitForStationReached(Port);
```

```
function [m, b] = GetAngleCycleCoeffs()

% This function is the only place in which these coefficients are defined.
% It exists to simplify changes in calibration because these coefficients are
% used in more than one place but always retrieved from here.

m = 15.5555555;
b = -0.28888888;
```

```
function [m, b] = GetDistanceCycleCoeffs()

% This function is the only place in which these coefficients are defined.
% It exists to simplify changes in calibration because these coefficients are
% used in more than one place but always retrieved from here.

m = 6.765;
b = -1.96;
```

```
function [NewLoc, NewDir, Error] = GoToLocation(Port, x, y, CurrentLoc, CurrentDir, LastStation)

% This function prepares to tell the robot to go to location x,y

% Get the new location from the Stations array
NewLoc = [x, y];
NewDir = CurrentDir;
Error = 0;

% Make sure that the provided x,y are not the coordinates of the current position.
arg1 = ourround(CurrentLoc(1),1) ~= ourround(NewLoc(1),1);
arg2 = ourround(CurrentLoc(2),1) ~= ourround(NewLoc(2),1);

if(arg1 | arg2)

        % Let the user know that a request has been received.
        disp(sprintf('Calculating move to location: %f, %f', x, y));

        % Compute the move amounts and the new direction
        [AnglCycles, DistCycles, TurnDir, NewDir] = DirectionCalc(NewLoc, CurrentLoc, CurrentDir);

        % Send the data to the Basic stamp to execute the computed move.
        Error = ExecuteMove(Port, AnglCycles, DistCycles, TurnDir,LastStation);

        % Use the actual cycle data to estimate the final location and direction of
        % the bot as accurately as possible.  This is done to try to help account for
    % numerical roundoff.

    [m, b] = GetAngleCycleCoeffs;
        if(round(AnglCycles) == 0)
         ActualAngle = 0;
        else
         ActualAngle = (AnglCycles - b) / m;
        end

        % compute actual nextd used by bot based on actual angle
        if(TurnDir == 0)
         ActualAngle = -ActualAngle;
        end

        rmat = [cos(ActualAngle), -sin(ActualAngle); sin(ActualAngle), cos(ActualAngle)];
        NewDir = (rmat * (CurrentDir./norm(CurrentDir))')';

        % compute actual distance in same fashion.

    [m, b] = GetDistanceCycleCoeffs;
        if(round(DistCycles) == 0)
         ActualDistance = 0;
        else
         ActualDistance = (DistCycles - b) / m;
        end

    % the most accurate NewLoc we can figure is the actual distance in the new direction + the current position.
        NewLoc = ActualDistance*NewDir + CurrentLoc;
end
```

```
function [NewLoc, NewDir, Error] = GoToStation(Port, Station, CurrentLoc, CurrentDir, LastStation)

% This function keeps track of where each station is in world coordinates.  It uses that information
% to pass along the command to travel to a particular station.

NewLoc = CurrentLoc;
NewDir = CurrentDir;
Error = 0;

% make sure that a valid station has been input.
if((Station > 8) | (Station < 1))
    disp(sprintf('Invalid station (%i) passed into GoToStation function.', Station));
    Error = 1;
else
       Stations = [1, 0; cos(pi/4), cos(pi/4); 0, 1; -cos(pi/4), cos(pi/4); -1, 0; -cos(pi/4), -cos(pi/4); 0, -
1; cos(pi/4), -cos(pi/4)] * 24;
    [NewLoc, NewDir, Error] = GoToLocation(Port,Stations(Station, 1), Stations(Station, 2), CurrentLoc,
CurrentDir,LastStation);
end
```

```
function res = ourround(X, ndp)

% This function rounds each element of X to ndp decimal points.  It is useful to
% reduce the problems of numerical roundoff.  We can compare two x,y sets without
% worrying about differences of hundredths of an inch for example.

res = zeros(length(X));

for i=1:length(X)
    if(X(i) > 0)
        res(i) = floor((X(i) * 10^ndp)+0.5) / 10^ndp;
    else
        res(i) = ceil((X(i) * 10^ndp)-0.5) / 10^ndp;
    end
end
```

```
function [x, y] = RetrieveLocation()

% This function is used to retrieve x,y coordinates from the user in the case of
% mode 3 which is "User Input Destination".


% get the x coordinate.
disp(' ');
disp('Enter the x coordinate of the desired location (-100 to 100):');

chosen = 0;
valid = 0;
x = 0;
y = 0;

% loop until a valid input is received
while(valid == 0)

    chosen = input(': ');
    if((chosen > 100) | (chosen < -100))
        disp('Please enter a value between -100 and 100.');
    else
        x = chosen;
        valid = 1;
    end
end


% get the y coordinate.
disp(' ');
disp('Enter the y coordinate of the desired location (-100 to 100):');

chosen = 0;
valid = 0;

% again, loop until a valid input is received
while(valid == 0)

    chosen = input(': ');
    if((chosen > 100) | (chosen < -100))
        disp('Please enter a value between -100 and 100.');
    else
        y = chosen;
        valid = 1;
    end
end
```

```
function Stations = RetrieveStations()

% This function takes in any number of stations to be part of a circuit.
% It checks for valid inputs and stores and returns them in Stations.

disp(' ');
disp('Enter station numbers (1-8) to be traversed.  Enter a 0 when done.');

chosen = 9;
stationct = 1;
Stations = 0;

% keep accepting stations until a 0 is entered.
while(chosen ~= 0)

    chosen = input(': ');
    if((chosen > 8) | (chosen < 0))
        disp('Valid station entries are 1-8 only.');
    elseif(chosen > 0)
        Stations(stationct) = chosen;
        stationct = stationct + 1;
    end
end
```

```
function [NewLoc, NewDir, Error] = TraverseStations(Port, Stations, InitialLoc, InitialDir)

% This function is used to send along the stations from Stations one at a time.

NewLoc = InitialLoc;
NewDir = InitialDir;
Error = 0;

% Do for each station in Stations.
for i=1:length(Stations)

    % issue the station change command.
    LastStation = i==length(Stations);

    % Issue the GoToStation for the current station.
    [NewLoc, NewDir, Error] = GoToStation(Port, Stations(i), NewLoc, NewDir, LastStation);

    % Check to see that the station change completed successfully.
    if(Error ~= 0)
        disp(sprintf('Station %i was not reached, aborting.', Stations(i)));
        break;
    end

end
```

```
function Station = WaitForStation(Port)

% This function waits for word from the stampworks kit indicating the next station to compute directions for.
disp('Awaiting Station Change...');

% Communication should be the id of the next station or an error.
Station = fscanf(Port, '%i');
```

```
function Error = WaitForStationReached(Port)

% This function waits for Port.Timeout for word that the robot has reached its destination.
% If word is not recieved, the user has the option to wait longer or abort.

disp('Waiting For Move Completion...');

Continue = 1;
Error = 0;
while(Continue == 1)

    % Wait for the reply.
    [Error,COUNT,MSG] = fscanf(Port, '%i');

    if((Error ~= 10) & (Error ~= 11))
        disp('Illegitimate message received while waiting for bot response.');
    end
    Error = Error  - 10;

    % if fscanf fails (like with a timeout), MSG will contain the error.  Show it and offer options.
    if(length(MSG) ~= 0)
        disp('Confirmation for station reached not received from the Bot.  The following message was
returned:');
        disp(' ');
        disp(MSG);
        disp(' ');

        disp('How would you like to proceed:');
        disp('    1: Try again');
               disp('    2: Abort');
               ToDo = input(': ');

        switch ToDo
            case 1,
                disp('Trying Again...');
                Continue = 1;
            case 2,
                Error = 1;
                Continue = 0;
            otherwise
                Error = 1;
                Continue = 0;
        end
    else
        Continue = 0;
       end
end
```

Appendix II: PBasic Source Code For Robot

```
'{$STAMP BS2}
'{$PORT COM2}


RIGHT   CON     0
LEFT    CON     1

RightServoPin  CON     13
LeftServoPin   CON     12

Direction      VAR     Bit
AnglCycles     VAR     Word
DistCycles     VAR     Word

DoneMoving             CON     1
InstructionsReceived   CON     2

Index   VAR     Word

BAUDRF  CON     16780


' Do leading initialization
GOSUB DoInitializaiton


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code continuously checks for instructions from
' the base station.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Main:
        GOSUB WaitForInstruction
        GOSUB ReportInstructionReceived
        GOSUB DoRotation
        GOSUB DoMoveForward
        GOSUB ReportFinishedMoving
        GOTO Main


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code continuously checks in 1 second increments
' for instructions to be recieved.  Before each check, it sends
' a message that it is waiting for instructions just in case the
' base is waiting for that message.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
WaitForInstruction:
        SERIN 0,BAUDRF, [WAIT("A"), Direction, AnglCycles.LowByte, AnglCycles.HighByte, DistCycles.LowByte,
DistCycles.HighByte]   ' Wait for ASCII letter A
        PAUSE 50
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code executes a rotation in the specified direction
' for the specified number of cycles.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
DoRotation:
        ' output the number of cycles indicated by AnglCycles in Direction
        IF(AnglCycles = 0) THEN FinishRotate
        IF(Direction = LEFT) THEN RotateLeft
        RotateRight:
                FOR Index = 1 TO AnglCycles
                        PULSOUT LeftServoPin, 1000
                        PULSOUT RightServoPin, 1000
                        PAUSE 20
                NEXT
                PAUSE 500
                GOTO FinishRotate
```

```
        RotateLeft:
                FOR Index = 1 TO AnglCycles
                        PULSOUT LeftServoPin, 500
                        PULSOUT RightServoPin, 500
                        PAUSE 20
                NEXT
                PAUSE 500
        FinishRotate:
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code executes a forward move for the specified
' number of cycles.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
DoMoveForward:
        FOR Index = 1 TO DistCycles
                PULSOUT LeftServoPin, 908
                PULSOUT RightServoPin, 500
                PAUSE 20
        NEXT
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code sends the message that instuctions have been
' received.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
ReportInstructionReceived:
        PAUSE 200
        SEROUT 1,BAUDRF,["B",InstructionsReceived]
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code sends the message that no moving is going on
' right now (the robot is idle).
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
ReportFinishedMoving:
        PAUSE 100
        SEROUT 1,BAUDRF,["B",DoneMoving]
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code does the initializaiton that is necessary
' at program startup.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
DoInitializaiton:
        DIRS = %1111111111111110
        Low LeftServoPin
        Low RightServoPin
        Index = 0
RETURN
```

Appendix III: PBasic Source Code For Robot

```
'{$STAMP BS2}
'{$PORT COM2}

Waiting        CON     9

BotIsWaiting   VAR     Bit

FALSE   CON     0
TRUE    CON     1

Baud48  CON     188
RxD     CON     12
TxD     CON     13

RIGHT   CON     0
LEFT    CON     1

FromBotPin     CON     14
ToBotPin       CON     15

Station1       CON     1
Station2       CON     2
Station3       CON     3
Station4       CON     4
Station5       CON     5
Station6       CON     6
Station7       CON     7
Station8       CON     8

NoBot          CON     11
StationReached CON     10

ReadVariable   VAR     Word

BotResponse    VAR     Nib

Counter        VAR     Nib
NextStation    VAR     Nib

LastStation    VAR     Bit
Direction      VAR     Bit
AnglCycles     VAR     Word
DistCycles     VAR     Word

Button0Trigger VAR     Word
Button1Trigger VAR     Word
Button2Trigger VAR     Word
Button3Trigger VAR     Word
Button4Trigger VAR     Word
Button5Trigger VAR     Word
Button6Trigger VAR     Word
Button7Trigger VAR     Word

BAUDRF  CON     16780


' Do leading initialization
GOSUB DoInitialization


' Find out which mode of operation to use
GOSUB ReadMatlabInput

' The user will input desired stations one at a time.
IF(ReadVariable = 1) THEN MoveRealTime

' The user will put in a circuit of stations to be traversed
IF(ReadVariable = 2) THEN MovePreDefined
```

```
' The user will specify a custom x, y location to go to.
IF(ReadVariable = 3) THEN MoveUserLocation
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code continuously checks for button hits in order
' to facilitate the real time station selection mode.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
MoveRealTime:
        IF(BotIsWaiting = TRUE) THEN SkipNextLine_2
        GOSUB WaitForBotFinishMoveMessage
        SkipNextLine_2:
        GOTO CheckButtons
        ReturnFrom_CheckButtons:
        GOTO MoveRealTime


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code keeps looping and retrieving the next station
' from matlab until the last station flag is set to facilitate the
' circuit of stations mode.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
MovePreDefined:
        IF(BotIsWaiting = TRUE) THEN SkipNextLine_4
        GOSUB WaitForBotFinishMoveMessage
        SkipNextLine_4:
        IF (LastStation = TRUE) THEN FinishPreDefinedMoves
        GOSUB StationChange
        GOTO MovePreDefined

FinishPreDefinedMoves:
        LastStation = FALSE
        GOTO MovePreDefined


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code keeps looping and retrieving the next user
' input move change to facilitate the custom x, y location mode.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
MoveUserLocation:
        IF(BotIsWaiting = TRUE) THEN SkipNextLine_5
        GOSUB WaitForBotFinishMoveMessage
        SkipNextLine_5:
        GOSUB StationChange
        GOTO MoveUserLocation


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is used to when buttons get pressed.  When a
' button is pressed, it indicates time to move to a new station.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
CheckButtons:
        BUTTON 0, 0, 255, 255, Button0Trigger, 1, OnButton0
        ReturnFrom_OnButton0:

        BUTTON 1, 0, 255, 255, Button1Trigger, 1, OnButton1
        ReturnFrom_OnButton1:

        BUTTON 2, 0, 255, 255, Button2Trigger, 1, OnButton2
        ReturnFrom_OnButton2:

        BUTTON 3, 0, 255, 255, Button3Trigger, 1, OnButton3
        ReturnFrom_OnButton3:

        BUTTON 4, 0, 255, 255, Button4Trigger, 1, OnButton4
        ReturnFrom_OnButton4:

        BUTTON 5, 0, 255, 255, Button5Trigger, 1, OnButton5
        ReturnFrom_OnButton5:

        BUTTON 6, 0, 255, 255, Button6Trigger, 1, OnButton6
        ReturnFrom_OnButton6:
```

```
        BUTTON 7, 0, 255, 255, Button7Trigger, 1, OnButton7
        ReturnFrom_OnButton7:
GOTO ReturnFrom_CheckButtons


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 0.
' It causes movement to station 1.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton0:
        NextStation = Station1
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton0


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 1.
' It causes movement to station 2.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton1:
        NextStation = Station2
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton1


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 2.
' It causes movement to station 3.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton2:
        NextStation = Station3
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton2


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 3.
' It causes movement to station 4.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton3:
        NextStation = Station4
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton3


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 4.
' It causes movement to station 5.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton4:
        NextStation = Station5
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton4


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 5.
' It causes movement to station 6.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton5:
        NextStation = Station6
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton5


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 6.
' It causes movement to station 7.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton6:
```

```
        NextStation = Station7
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton6
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This block of code is called in response to a press on button 7.
' It causes movement to station 8.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnButton7:
        NextStation = Station8
        GOSUB OnStationChange
        GOTO ReturnFrom_OnButton7


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This label is used when the station change is initiated from
' the stampworks kit.  It tells Matlab which station to compute.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
OnStationChange:
        ' Send the new desired station info to matlab for directional processing
        PAUSE 100
        SEROUT TxD, Baud48, [DEC NextStation, 10]


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This label is used to retrieve the moving instructions computed
' by Matlab.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
StationChange:
        ' Get the results from matlab
        GOSUB ReadMatlabInput
        LastStation = ReadVariable

        GOSUB ReadMatlabInput
        Direction = ReadVariable

        GOSUB ReadMatlabInput
        AnglCycles = ReadVariable

        GOSUB ReadMatlabInput
        DistCycles = ReadVariable


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is where the instructions are sent to the bot.  This may
' be called for the initial send or for resend operations.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
ResendMovingInstruction:


        ' Send the new instructions to the robot.
        GOSUB SendMovingInstructionToBot

        ' Wait for verification that bot has reached its destination
        GOSUB WaitForBotReceiveMessage
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code reads in a single word sized variable from
' Matlab which will be sending it in 2 byte sized chunks.  The
' pause is necessary b/c matlab is actually sending a word and so
' it must be given time to finish sending the remaining 8 bits.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
ReadMatlabInput:
        ReadVariable = 0
        SERIN RxD,  Baud48, [ReadVariable.lowbyte]
        Pause 10
        SERIN RxD,  Baud48, [ReadVariable.highbyte]
        PAUSE 100
        SEROUT TxD, Baud48, [DEC TRUE, 10]
RETURN
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code is called if SERIN times out while waiting
' for the bot to respond that it received its instructions.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
MoveMessageNotReceived:
        DEBUG "Not Received..", CR
        Counter = Counter + 1
        IF(Counter < 3) THEN SkipNextLine_1
        BotIsWaiting = TRUE
        GOTO BotNotResponding
        SkipNextLine_1:
        DEBUG "Sending Again..", CR
        GOTO ResendMovingInstruction


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code is called if SERIN times out while waiting
' for the bot to respond that it has finished moving.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
DoneMessageNotReceived:
        DEBUG "Not Received..", CR
        Counter = Counter + 1
        IF(Counter < 3) THEN SkipNextLine_3
        BotIsWaiting = TRUE
        GOTO BotNotResponding
        SkipNextLine_3:
        DEBUG "Waiting Again..", CR
        GOSUB WaitForBotFinishMoveMessage


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code sends the 3 pieces of moving information
' necessary for movement to the bot.  The required info is:
'    Direction - Turn left or right
'    AnglCycles - How much to turn
'    DistCycles - How far to go forward
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
SendMovingInstructionToBot:
        DEBUG "Sending Move Instruction..", CR
        PAUSE 100
        SEROUT ToBotPin,BAUDRF,[TRUE,"A",Direction, AnglCycles.LowByte, AnglCycles.HighByte, DistCycles.LowByte,
DistCycles.HighByte]
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code waits for 60 seconds for the bot to finish
' moving and respond as such.  It will time out once (after 30
' seconds) before an error state is entered.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
WaitForBotFinishMoveMessage:
        DEBUG "Waiting For Finish Moving Message.."
        SERIN FromBotPin,BAUDRF,30000, DoneMessageNotReceived, [WAIT("B"), BotResponse]
        DEBUG "Received..", CR
        Counter = 0
        BotIsWaiting = TRUE
        PAUSE 100
        SEROUT TxD, Baud48, [DEC StationReached, 10]
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code waits for 6 seconds for the bot to respond
' that it has received its instructions.  It will time out twice
' (every 2 seconds) before an error state is entered.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
WaitForBotReceiveMessage:
        DEBUG "Waiting For Moving Instruction Verification.."
```

```
        SERIN FromBotPin,BAUDRF,2000, MoveMessageNotReceived, [WAIT("B"), BotResponse]
        DEBUG "Received..", CR
        Counter = 0
        BotIsWaiting = FALSE
RETURN


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code is called if the bot does not respond properly
' within the specified amount of time for an action.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
BotNotResponding:
        DEBUG "Bot is not responding..", CR
        PAUSE 100
        SEROUT TxD, Baud48, [DEC NoBot, 10]
        Counter = 0
        GOTO MoveRealTime


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' This is block of code does the initializaiton that is necessary
' at program startup.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
DoInitialization:
        DIRS = %1010111100000000
        DEBUG "Establishing Communication With Matlab...", CR
        PAUSE 100
        SEROUT TxD, Baud48, [DEC Waiting, 10]
        NextStation = Station1
        BotIsWaiting = TRUE
        Counter = 0
        LastStation = FALSE
RETURN
```