University at Buffalo

Department of Mechanical and Aerospace Engineering
MAE 576: Mechatronics
Spring 2003

Project Report

Temperature Sensing Robot

Submitted by:
Rajendra Agrawal
Govindarajan K Srimathveeravalli
Rageesh J Britto
William J Mitchell
Gopikrishnan Sidrardhan

Index of Contents

Abstract

The aim of this project is to create a mobot for remote operation, which either can function autonomously or can be controlled by a human operator. The mobot is built with two way communication capability. The communication from base station to the mobot is achieved using RF communication and the feedback from the mobot to the base station is given through an IR communication channel. The mobot can be actively controlled in real time by the user who is located in the base station. The mobot is aimed to serve as the "hands and eyes" of the user in a potentially hostile environment where direct physical presence of the user is not possible. The mobot is calibrated to travel in fixed increments, and visual feedback as to the distance moved is made available to the user using the LCD. Hence by using it the user can actively map the terrain covered. Also, even when under direct control of the user, the mobot is built to actively detect and avoid collisions. In the autonomous mode the mobot will travel a preprogrammed path heading towards a specific direction and will actively avoid collisions. The additional feature of this mode is that the robot will keep track of the path it has traversed and then can retrace its path back to the base station. The mobot has a temperature sensor on board which can be operated either using the base station or whenever a collision occurs. This facility is aimed at providing the mobot with a means to detect human presence in the environment mapped.

Introduction

Even though the final product of this project is the mobot and its base station which acts as its remote controller, the aim of this project is verily different. Microprocessor based systems have become commonplace now in our lives. However, we would find very few such applications where currently only a single microprocessor is used; another fact is that most of the times necessity would arise for the microprocessor of one system to communicate with that of another. Based on these observations it can be concluded that it would be of great instructional value to understand the dynamics of working and communicating with multiple microprocessors in a given environment. Furthermore, it would be more interesting to understand about wireless communication rather than communication using wired means. The reason being that, most microprocessors in a given environment may not be readily accessible using wired means. The microprocessors in concern are the 2 Basic Stamps, one mounted on the mobot and the other serving as the base station. In addition, there are multiple modes of communication available for use in this scenario. It becomes necessary to evaluate the various modes of communication and determine the strengths and weaknesses of each one. This forms the actual motivation of the project, the building of the mobot and having fun in the process is just a windfall.

Objectives and Goals

The goals and objectives of this project can be split two fold. The first being the objectives where in something of definite value is learned by doing. The goals will chalk out the roadmap for the project which would eventually transcend into something concrete which can be marketed as a concept or as a product.

The Objectives of the project are:

- Establish distributed sensing and control through wireless media between the mobile robot and the base station.
- Coupling using wired and wireless means. Use of IR and RF channel for communication.
- Creation of intelligent remote sensor using the mobot as the basis.
- Create remote control over the mobot.
- Integration of multiple processors in a given environment.
- Understand use of SERIN and SEROUT command.

The tangible goals of this project are:

- Creation of a mobot whose motion is controlled by the interfaced buttons in the base station.
- Use the mobot as a remote sensor to gather temperature data of the given environment.
- Make the mobot autonomous and using the onboard IR sensor enables collision detection. Make it intelligent to head for a particular direction and trace its path back to the base station.
- Enable two-way communication, RF to send commands to the mobot and use the IR set up to provide feedback to the base station.
- Use the LCD interface as a means of getting visual data from the robot, will help enable to map the path of the robot.
- Calibration of the mobot to move in predetermined steps. Implementation of open loop control.
- Option of Pre-Programming of the mobot to move along a set path or curve.

List of Components and Cost Estimate

Serial #	Name of the Component	Quantity Used	Price
1	Basic Stamp II Kit	1	the Kit
2	Basic Stamp BOE Kit	1	(Included in the kit)
3	IR Transmitter (Firestick) / Receiver	1 of each type	(Included in the kit)
4	RF Transmitter / Receiver	1 of each type	(Included in the kit)
5	Antenna for RF communication	2	(Included in the kit)
6	Matrix Orbital LCD, LCD1621	1	(Included in the kit)
7	IR LED	2	(Included in the kit)
8 9	IR Photo Detector Various Resistors	2 4	(Included in the kit) (Included in the kit)
10	Extra Breadboard	1	(Included in the kit)
11	DS 1620	1	(Included in the kit)
12	1.5 v Batteries	4	$4 (bought 8 batteries for the whole project)
13	9 v Batteries	1	$4 bought one for the entire project
14	BASIC Stamp II module	1	(Included in the kit)
15	300 mA 9 VDC power supply	1	(Included in the kit)
16	Serial cable	1	(Included in the kit)
17	infrared receiver (Panasonic PNA4602M)	2	(Included in the kit)
18	infrared LEDs covered with heat shrink tubing (QT QEC113)	2	(Included in the kit)
19	4-40 x 3/8”machine screws	8	(Included in the kit)
20	1" polyethylene ball, pre-drilled	1	(Included in the kit)
21	o-ring tires	2	(Included in the kit)
22	Boe-Bot plastic machined wheels	2	(Included in the kit)

23	4-40 x 3/8” flathead machine screws	2	(Included in the kit)
24	Boe-Bot aluminum chassis	1	(Included in the kit)
25	1/16" x 1.5” long cotter pin	1	(Included in the kit)
26	4-40 locknuts	10	(Included in the kit)
27	13/32" rubber grommet (fits ½” hole)	1	(Included in the kit)
28	9/32" rubber grommet (fits 3/8” hole)	2	(Included in the kit)
29	Battery holder with cable and barrel plug	1	(Included in the kit)
30	Parallax pre-modified servos	2	(Included in the kit)

The whole project was assembled and executed using the equipment given in the kit itself. Additional expense of any sort was avoided. For commercial implementation of any product, care must be taken to maximize the cost to utility difference. That is for minimum cost maximum utility must be extracted from the device.

Component & Circuit Description

As any other application built so far, this set up also consists of many individual circuits and components. The various circuits used in the set up are:

- Communication Set up, IR and RF circuits.
- Set up of the Base Station.
- Mobot set up.
- Collision Detection.
- Temperature Sensing.
- LCD circuit

RF communication:

RF communication forms the primary means of communication in this set up and helps the passage of commands from the base station to the mobot. The primary advantages of using a RF communication set up is linked to the ease of use, the wide area of communication (unlike the IR there is no need for line of sight here), the range (over 100 feet) and it does not need a dedicated power source (unlike the IR which ate up a 9v battery).

The downside of using a RF set up for communication is that it is preferable to have encoding to enable safe and complete transit of data.

The RF circuit consists of two sub-circuits, one is for the receiver and the other is for the transmitter. In this project, the transmitter is mounted on the base station and the receiver is on the mobot.

The set up of the receiver is as follows:

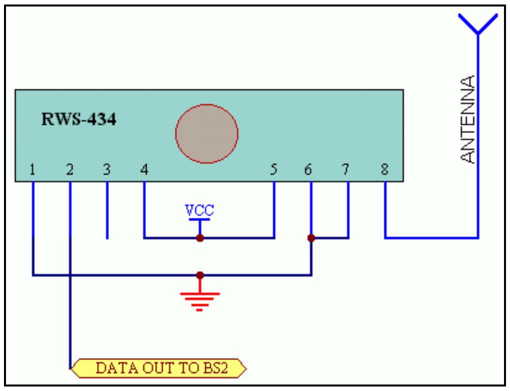


Figure 1 RF receiver, Source: http://www.rentron.com/Stamp_RF.htm

The above diagram is a general representation of the circuit. Here there is major deviation from the diagram that instead of pin 2, pin 3 is connected to the Basic Stamp through a resistance. The key difference is that the pin number 2 is said to be used for digital data output and the pin number 3 is used for linear data output. It is connected to pin number 2 only if connected through an encoder set up, otherwise pin number 3 is preferred. The rest of the circuit is connected in the same manner. In conjunction with this circuit, we use the SERIN command to take the incoming signal from the transmitter on the base. The set up also requires an antenna each on both ends.

The other end of this circuit is a transmitter. It is mounted on the base station. The base station in this case is interfaced with a number of buttons. Upon pressing each button a different message is released using the SEROUT command, and based on the message sent a particular operation is carried out by the mobot. Given below is a general representation of the transmitter.

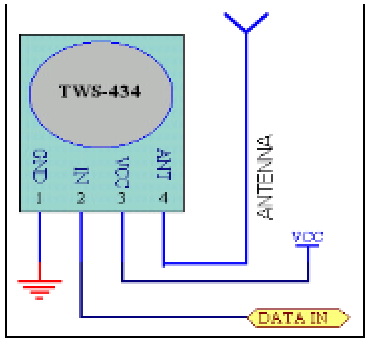


Figure 2 RF transmitter, Source: http://www.rentron.com/Stamp_RF.htm

There is again a small error associated with the above diagram, the picture above is only representative and does not match with the actual component used in the base station. However, the circuit connections remain the same. Pin 2 is used to link the transmitter with the basic stamp and the rest of the pins are connected as shown in the diagram. The signal from the transmitter is used SEROUT command. The connection itself is set up on par at 1200 baud.

IR communication

The IR communication is set up using the given a Fire-Stick and Receiver for the IR signal. In the given scenario, it is desired that we receive proper feedback or receive data from the mobot that it has sensed/detected. To achieve the same the IR communication set up is used. IR set up though commonly used has some inherent problems, like line of sight, use of a separate power source and noise due to other thermal sources. Inspite of these problems the IR set up is commonly used in many applications including the common TV remote. Here again the SERIN and SEROUT commands used to enable the communication to occur.

As mentioned before the transmitter of the IR communication set up is mounted on the robot. A representational diagram of the same is as given below. The actual circuit on the mobot is connected in a similar fashion, though the middle pin is left free and is not connected to anything as such. The Firestick gave a very good response and functioned well even if the receiver/sensor was placed at 180^0. Additionally the range of the sensor was around 80 feet and the time lag in giving feedback was not large.

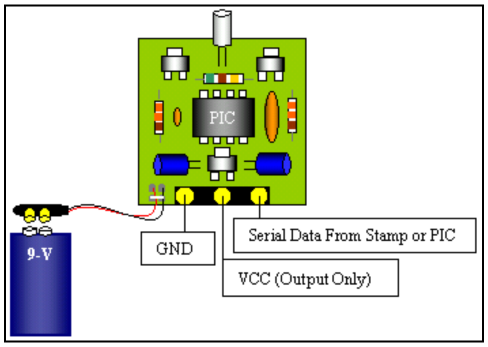


Figure 3 IR transmitter, Source: http://www.rentron.com/FS-2.htm

The IR receiver was mounted on the base station. A representational circuit diagram of the same is given below. The actual pin set up is also as given in the diagram.

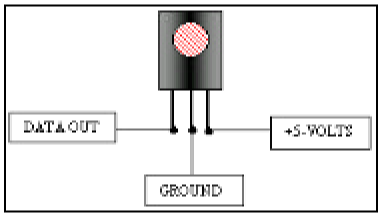


Figure 4 IR Receiver: http://www.rentron.com/FS-2.htm

Set up of the Base Station

The base station serves three purposes in this project.

1. It serves as the remote control station for the mobot.
2. It accepts the incoming data from the mobot.
3. It displays the status of the mobot and helps determine the path taken by it.
4. It displays the temperature of the environment as taken by the robot.

For the remote controlling of the mobot, buttons D0 to D4 and D5 are interfaced with pins 1 to 5 and pin 7 respectively. The entire scene is orchestrated with the use of the button command and when a particular button is pressed, a corresponding SEROUT command is send having a specific message. Based on the message contained in the SEROUT command the mobot carries out some specified task.

Pin numbers 0 and 6 are used to interface the RF transmitter and the IR receiver respectively. The buttons interfaced in the base station serve as means for remote control, the pins interfaced with the RF transmitter to send data and the IR receiver for getting incoming temperature and positional data.

Pin numbers 15 through 10 are used to interface the LCD circuit and serve to display the distance and direction of the mobot's motion.

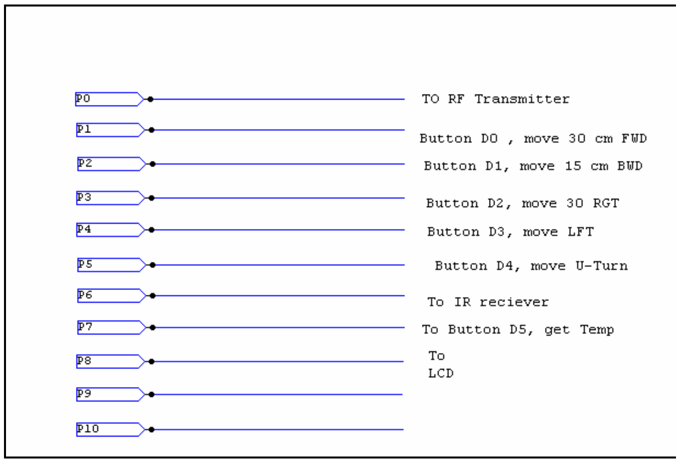


Figure 5 Basic Circuit Set Up of the Base Station

Mobot Setup

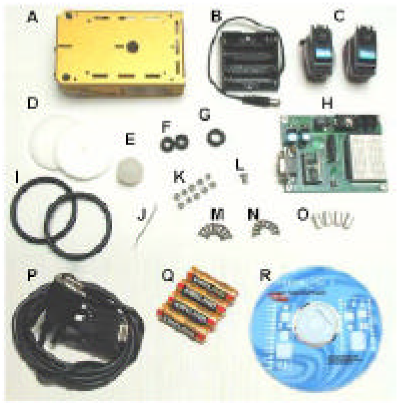


Figure 6 Boe-Bot Parts

The first major task was to get the Boe-Bot assembled and running properly. A robot's subsystems include its motors, sensor arrays, microprocessor, and mechanical linkages. These have to be assembled into a working system. Next we test and trouble-shoot the subsystems. Then comes the system integration, the process of making all the Robot's subsystems work together. Once the testing and trouble-shooting is finished at the subsystem level, a robot's subsystems have to be connected to and controlled by a microprocessor. The process of getting all the subsystems (including the microprocessor) to work together to make the robot perform its assigned task list is called system integration. System integration can be tricky to begin with, but robotics teams who skipped any of the testing and troubleshooting at the subsystem level often have much larger problems with their system integration. That is why we spent some time trying to get the robot to work the way it's supposed to with all the bugs removed from the subsystem.

The Boe-Bot setup can be separated into the following activities:

1. Boe-Bot Mechanical Assembly
2. Programming the Boe-Bot's BASIC Stamp 2 On-Board Computer
3. Testing the Servos Individually
4. Running Both Servos
5. Tuning the Servos – Calibration in Software

Each of these activities involves discrete steps to get the Boe-Bot up and running. First, we had to check to make sure that we had all our parts. We found that a whisker and two screws for the servos was missing and had to get them replaced by the TA. Next, we put the mechanical parts together.

After that, we had to test the microprocessor subsystem. Then came the testing of each servo motor individually. Then, the servo motors had to be made to work in unison. Last, but certainly not least, calibrate the pre-modified servos. By carefully following the instructions in the Robotics manual, we ensured that our microprocessor and motor subsystems were working reliably.

Collision detection

The Boe-Bot uses infrared LEDs like headlights for object detection. They emit infrared, and in some cases, the infrared reflects off objects, and bounces back in the direction of the Boe-Bot. The eyes of the Boe-Bot are the infrared detectors. The inf rared detectors send signals to the BASIC Stamp indicating whether or not they detect infrared reflected off an object. The brain of the Boe-Bot, the BASIC Stamp, makes decisions and operates the servo motors based on this input.

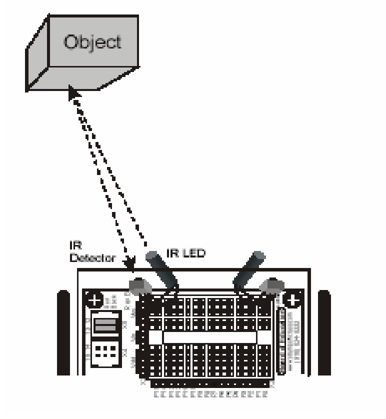


Figure 7 Object Detection

The IR detectors have built-in optical filters that allow very little light except the 980 nm. infrared that we want to detect onto its internal photodiode sensor. The infrared detector also has an electronic filter that only allows signals around 38.5 kHz to pass through. In other words, the detector is only looking for infrared flashed on and off at 38,500 times per second. This prevents interference from common IR interference sources such as sunlight and indoor lighting. Sunlight is DC interference (0 Hz), and house lighting tends to flash on and off at either 100 or 120 Hz, depending on the main power source in the country where you reside. Since 120 Hz is way outside the electronic filter’s 38.5 kHz band pass frequency, it is, for all practical purposes, completely ignored by the IR detectors.

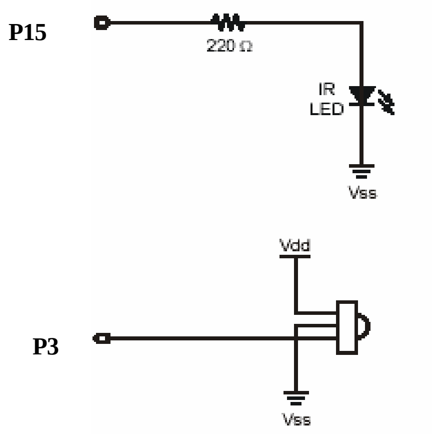


Figure 8 Infra-Red Circuit

How the IR Pairs Display Program Works

Two bit variables are declared to store the value of each IR detector output. The command freqout 15, 1, 38500 sends the on-off pattern left IR LED circuit by causing it to flash on and off rapidly. The harmonic contained in this signal either bounces off an object, or not. If it bounces off an object and is seen by the IR detector, the IR detector sends a low signal to IO pin P14. Otherwise, the IR detector sends a high signal to P14. So long as the next command after the freqout command is the one testing the state of the IR detector's output, it can be saved as a variable value in RAM. The statement left_IR_det = in14 checks P14, and saves the value ("1" for high or "0" for low) in the left_IR_det bit variable. This process is repeated for the other IR pair, and the IR detector's output is saved in the right_IR_det variable.

The saved bit values for each IR detector output can be used to control the motion of the Boe-Bot in the required manner by using the following code:

```
if left_IR_det = 0 and right_IR_det = 0 then u_turn
if left_IR_det = 0 then right_turn
if right_IR_det = 0 then left_turn
```

Temperature sensing

This part uses the Dallas Semiconductor DS1620 digital thermometer/thermostat chip. This chip measures temperature and makes it available to the BASIC Stamp through a synchronous serial interface. The DS1620 is an intelligent device and, once programmed, is capable of stand-alone operation using the T(com), T(hi) and T(lo) outputs. The DS1620 requires initialization before use. In active applications like this, the DS1620 is configured for free running with a CPU. After the configuration data is sent to the DS1620, a delay of 10 milliseconds is required so that the configuration can be written to the DS1620's internal EEPROM. After the delay, the DS1620 is instructed to start continuous conversions. This will ensure a current temperature reading when the BASIC Stamp requests it. To retrieve the current temperature, the Read Temperature ($AA) command byte is sent to the DS1620. Then the latest conversion value is read back. The data returned is nine bits wide. Bit8 indicates the sign of the temperature. If negative (sign bit is 1), the other eight bits hold the two's compliment value of the temperature. Whether negative or positive, each bit of the temperature is equal to 0.5 degrees Celsius.

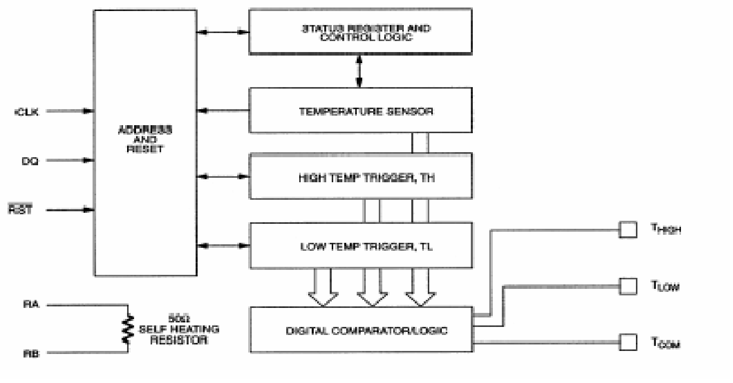


Figure 9 DS1620 FUNCTIONAL BLOCK DIAGRAM

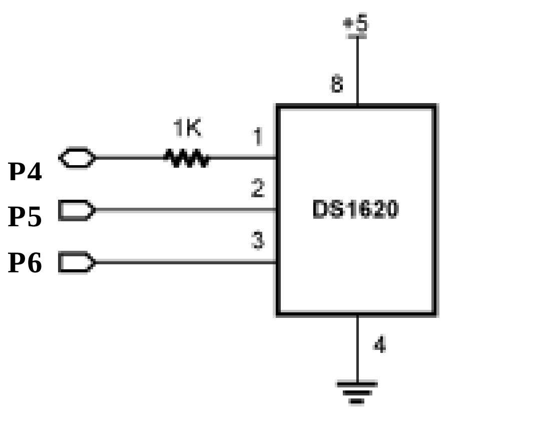


Figure 10 DS1620 Circuit

Application note:

There are two modes in which the mobot operates. The remote mode and the autonomous mode. The initial display is a welcome to the two modes and asks the user to select one from the list. From there one the following list can be followed to run the mobot application.

- Press button 7 or 8 to select between modes 1 and 2 respectively.
- If mode 1 is selected the robot will run by remote control.
- In mode 1, button 1 controls forward motion, with the robot moving in steps of 5 cms.
- Button 2 gives motion in backward direction with same step size as in the previous case.
- Button 3 and 4 are for left and right turns respectively. The turns are taken in 30 degree angles.
- Button 5 is for making the robot do a U-turn.
- Button 6 make the robot read the temperature of the surrounds and sends it back to the base station.
- Alternatively if mode 8 is selected the robot goes into its autonomous mode. Here the robot moves for 32 discrete motions, records the value in the EEPROM and then on completion of 32 steps retraces its path back to the base station.
- The system cannot go from mode 2 to mode 1 while mode 2 is running, but vice versa is possible.
- There is continuous visual feedback as to what the machine is doing at any instant through LCD.

Problems Faced:

1. The limitation of array size. We can't declare an array size greater than 32. So our robot can move maximum of 32 steps.

2. We were unable to create a proper temperature Sink or Source. So, our criteria for the robot to check the temperature and comes back to original position only when it hits something of high or low value than normal range, was not fulfilled.

3. SERIN and SEROUT commands. The problem is once you give a SEROUT command from a Micro-Processor then SERIN should be called just after it in another Micro-Processor. Synchronizing the 2 things is a difficult task. One way to come out of it is to make SERIN keep on waiting until it receives a signal. But this is not an efficient method, since this makes that Micro-Processor cannot do any other activity while this waiting. We aren't able to send the exact positions and temperatures back to the station all the times. One time we were getting the value of temperature as 24 degrees and one time 255 degree, for the same code after the robot hits the same object again.

4. The range of IR-LED depends on its position and orientation. This sensor is too sensitive, and so gives inaccurate results, depending on its position and the surface orientation of the object it detects.

5. Initially we noticed our Boe-Bot didn't go perfectly straight forward when we ran Program for forward motion. For that matter, it didn't go perfectly straight backward in response backward motion loop also.We found that the Boe-Bot veered to the right when it was programmed to go straight forward, therefore either the left wheel needs to slow down, or the right wheel needs to speed up. Since the servos are pretty close to top speed as it is, slowing the left wheel down will work better. This was done by making the pulse period to the left servo, which is connected to P13, smaller. By trying different values, we were able to home in on the values that made our Boe-Bot wheels turn in the direction we wanted.

   Given below is the final code for controlling the motion of the Boe-Bot with the calibrated values.

Forward:
```
for pulse_count = 1 to 20
      pulsout 12, 500
      pulsout 13, 920
      pause 20
next
```

left_turn:
```
for pulse_count = 0 to 9
      pulsout 12, 500
      pulsout 13, 580
      pause 20
next
```

right_turn:
```
      for pulse_count = 0 to 9
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next

u_turn:
      for pulse_count = 0 to 54
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next

backward:
      for pulse_count = 1 to 10
            pulsout 12, 1000
            pulsout 13, 580
            pause 20
      next
```

One of the issues during the calibration was that we found that the Boe -Bot had to be recalibrated every day. We finally pinned this down on the batteries. We found out that the speed of rotation of the wheels was dependant on the life of the batteries. We got around this problem by calibrating the Boe-Bot with a new set of batteries and then using them only during the final demonstration of the project.

New Commands Used

The two new commands used for this project were SERIN and SEROUT. These PBasic commands are actually complementary in nature. The SERIN command is used receive data that is sent out by any device, generally a microcontroller, in an asynchronous manner. The SEROUT command is used to send the data.

An example of the two commands in use is as illustrated below (from our source code).

```
SERIN 6, 813, [Msg]
SEROUT 0, 16780, ["1"]
```

The SERIN here is used to receive data sent out by the IR of the mobot. The SERIN command takes many parameters and can be learn about from the Basic Stamp manual. The SERIN that has been implemented above takes 3 parameters. The first parameter is the pin to which the receiving device is connected. In this case the receiving device is an IR detector/receiver. The second parameter is the baud rate at which data has to be transferred. The third parameter is the variable using which the incoming data is stored.

The SEROUT command as mentioned before is used to send out serial asynchronous data out. The SEROUT command shown above is activate or called when a particular button on the base station is pressed. The SEROUT command also takes in many parameters which can exploited, in our case the implementation takes just three parameters. The first parameter is the pin to which the transmitting device is connected to. In this case the output device is a RF transmitter. The second parameter is the baud rate and the final parameter is the bit of data transmitted. The bit of data here is used perform some meaningful function. The mobot upon receipt of that piece of data moves forward.

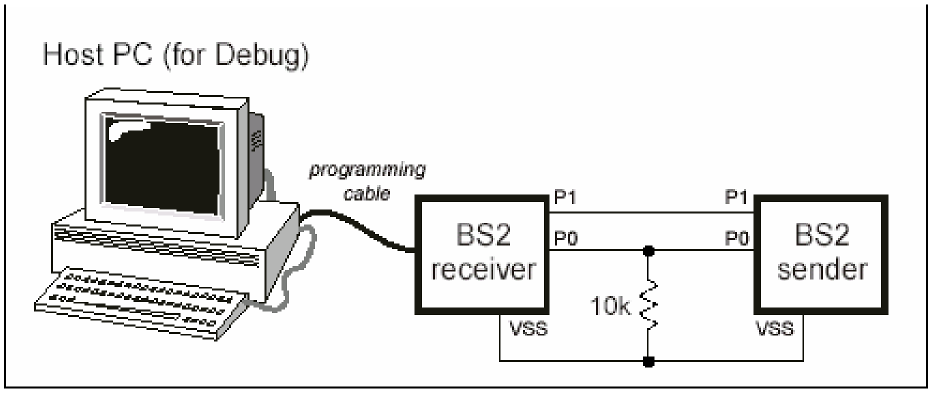


Figure 11 A schematic of the two basic stamp interfaced with each other through wire based asynchronous serial communication

Working of the Program

The base station program is relatively simple: it merely checks for the button events from the user and outputs the user command in an appropriate format to the mobile, and if there are no user commands, it looks for data from the mobile and outputs the relevant information on the LCD screen.The mobile program is a bit more sophisticated; first it tries to get the input from the base station. If there is any input, the first check it makes it is whether the input is button D6 or D7. In case it is D6, the mobile unit is switched to mode 1 (autonomous mode). If it is D7, it is switched to mode 2 (button mode).

In autonomous mode, the program checks for the current array index. If it is less than 32, the mobile unit moves forward, stores the movement it made in the array provided for the purpose and increments the array index by 1. Then, it checks if the sensors have detected a collision and takes the appropriate decision – if the sensors have detected a collision on both sides, a u-turn is made, or if a collision is detected on the left side, a right turn is made, or if a collision is detected on the right side, a left turn is made. Again, all the movements are recorded in the array.

If the array index is greater than or equal to 32, the mobile unit attempts to retrace its path. For this, the first step is a U-Turn, followed by traversing the array, taking the next move from it in the reverse order in which they were pushed into the array. This process continues until the array is empty, i.e., the array index is zero.

In the button mode the first step is to get the input from the base station using SERIN. Then, depending on the incoming command, various actions are performed. If the command corresponds to button D0 being pressed on the base station, impulses are given to the servos (which drive the wheels) so that the unit moves forward by a pre-defined distance. If the command is button D1, the unit moves back. Similarly, commands D2, D3, and D4 correspond to a left turn, a right turn and a U-turn. Button D5 is a request for the temperature to be taken. On receiving this command, the mobile unit takes the temperature and sends it back to the base unit using the SEROUT command. The D6 command causes the mobile unit to switch modes and enter the autonomous mode.

Final Result/Conclusions

1. We used all the 8 buttons provided in bread-board. D0, D1, D2, D3, are used for giving increments to the robot in forward, backward, right and left direction respectively. D4 is used for U-turn. D5 is used to take temperature of the object near to robot in current position. D6 and D7 are used for switching the modes as 1st mode or 2nd mode.

2. In the 1st mode we can control the robot using buttons to move in discrete steps of angles i.e. 30 degrees or distances i.e. 5cm.

3. In this mode, if the robot hits an object, it moves back to 2.5cm and sends the message to the base station, saying there is a collision.

4. After reaching a particular position and orientation, we can actuate the autonomous mode.

5. The robot moves by itself for 32 steps avoiding collisions, sensing all the temperatures of different objects, but unable to send the exact values sometimes to the base station.

6. After reaching to 32 steps, it sends the last temperature and returns to original position using the same path, as this is stored in an array with acceptable accuracy i.e. a range of 0-6cm difference in the actual location and the location it reaches afterwards.

7. Now the user can again change the orientation or position or both and starts mapping the temperature of new area.

8. The movement of robot in autonomous mode is depending on its collision on IRLED_LEFT or IRLED_RIGHT. If IRLED_LEFT detects a collision, the robot takes 30 Degrees RIGHT, and for IRLED_RIGHT it takes 30 Degrees LEFT. If both the IRLED detects collision simultaneously, then the robot takes a U-TURN.

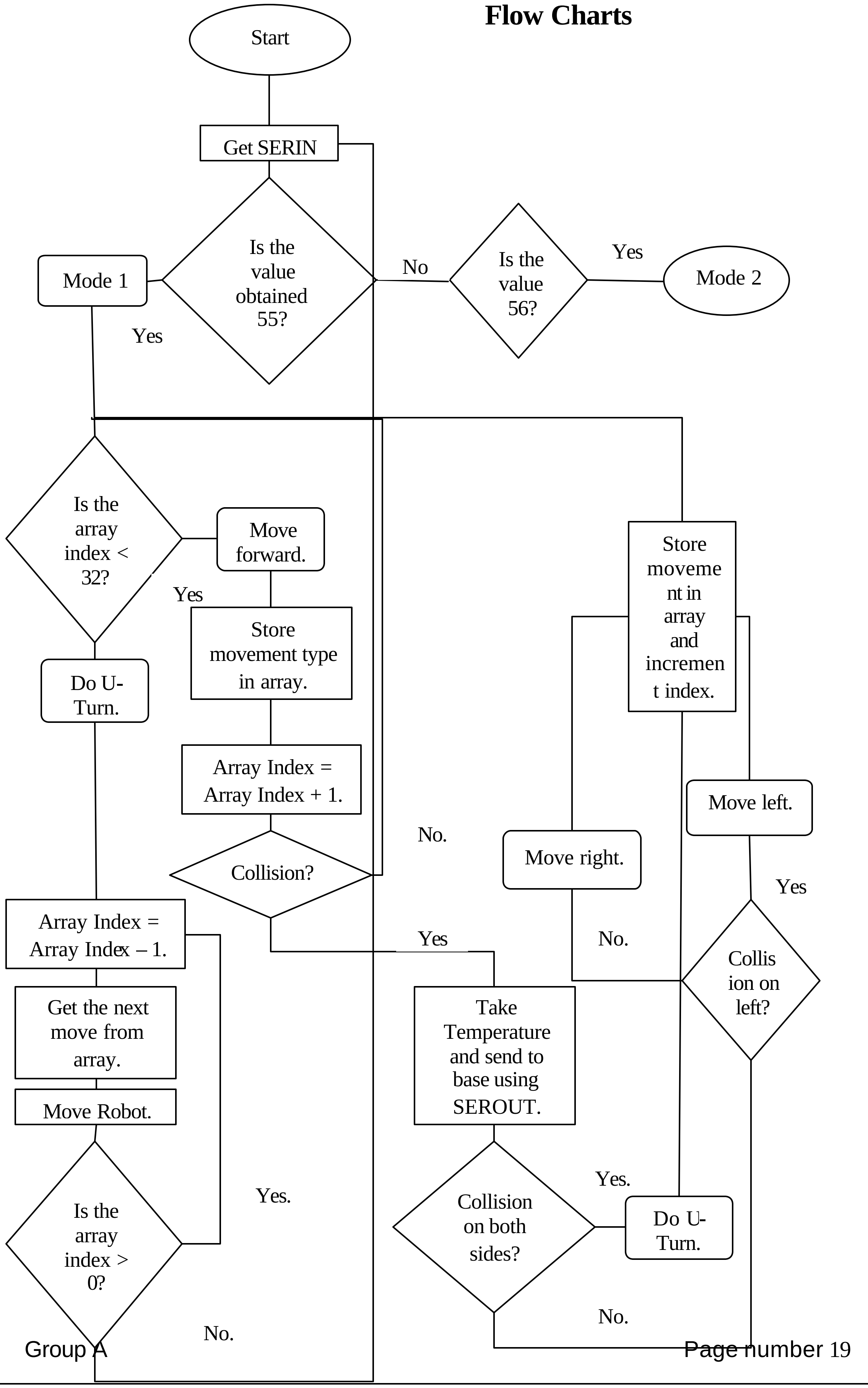
9. In the autonomous mode the mobot finds the path out in a given direction and will travel in that general direction for 32 steps. Then it would stop and return back to base. There is however no collision detection on the way back.

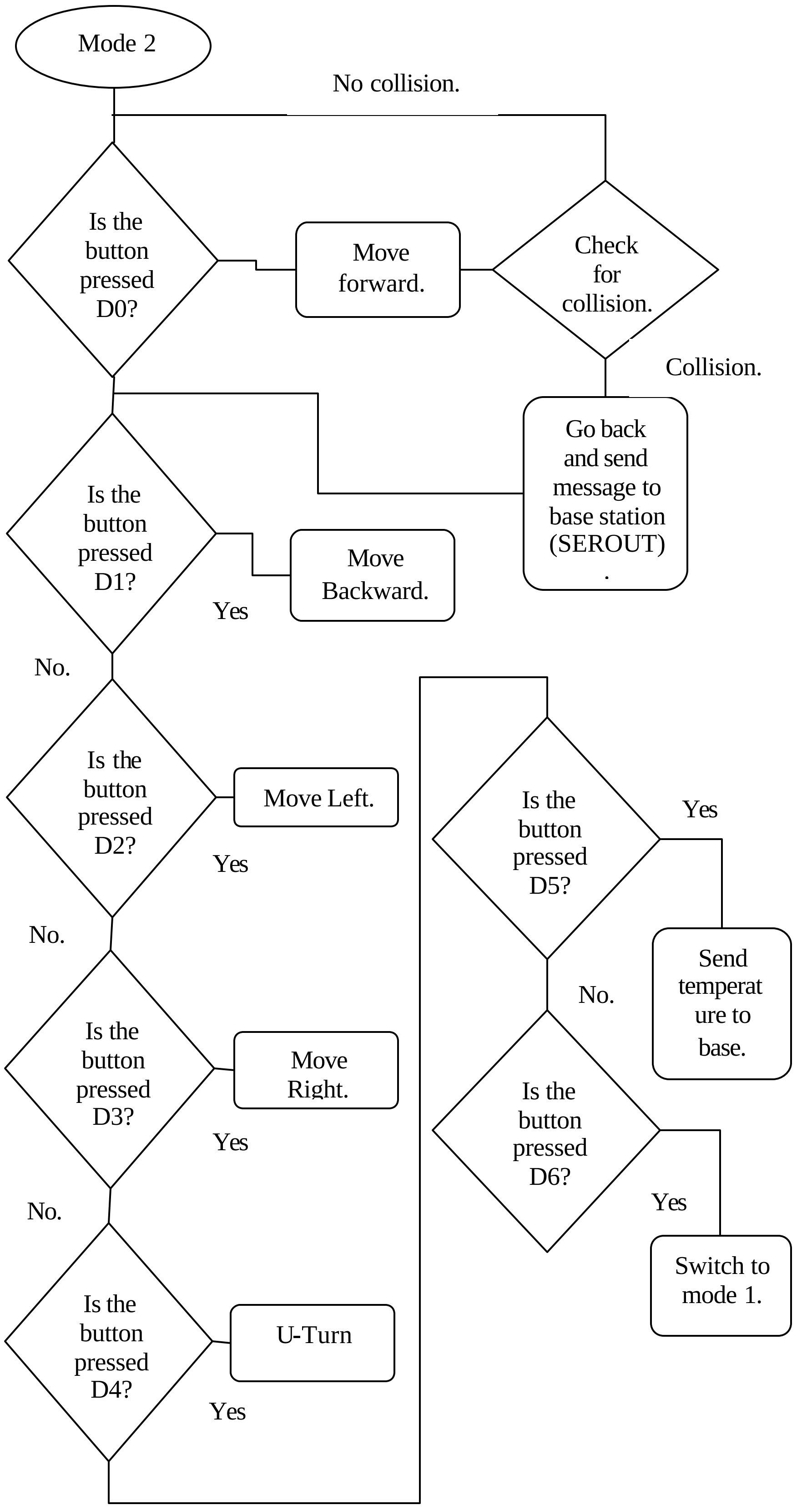
10. There is continuous visual feedback of the mobot’s movements through the LCD.

Contribution of Team members to Project

Person #	Name	Contribution
30314938	Rajendra Agarwal	Programming the base station, setting up the IR and RF communication, enabling the robot for autonomous mode, interfacing the robot with buttons for remote mode and documentation.
30412424	Regeesh Britto	Setting up of the robot calibration of the robot, setting up of the temperature sensor,. Collision detection, documentation
30510714	Gopikrishnan Sidhardhan	Calibration of the robot, setting up of the temperature sensor, Collision detection and documentation
	William Mitchell	Calibration of robot, collision detection and documentation.
30465741	Govindarajan Srimath Veeravalli	Programming the base station, setting up the IR and RF communication, enabling the robot for autonomous mode, interfacing the robot with buttons for remote mode and documentation.

Flow Charts


Start
Get SERIN
Is the value obtained 55?
Mode 1
Yes
No
Is the value 56?
Yes
Mode 2
Is the array index < 32?
Move forward.
Yes
Store movement type in array.
Do U-Turn.
Array Index = Array Index + 1.
Collision?
No.
Array Index = Array Index – 1.
Yes
Get the next move from array.
Move Robot.
Is the array index > 0?
Yes.
No.
Store movement in array and increment index.
Move right.
Move left.
Yes
No.
Collision on left?
Take Temperature and send to base using SEROUT.
Collision on both sides?
Yes.
Do U-Turn.
No.


Mode 2
No collision.
Is the button pressed D0?
Move forward.
Check for collision.
Collision.
Go back and send message to base station (SEROUT).
Is the button pressed D1?
Yes
Move Backward.
No.
Is the button pressed D2?
Move Left.
Yes
No.
Is the button pressed D3?
Move Right.
Yes
No.
Is the button pressed D4?
U-Turn
Yes
Is the button pressed D5?
Yes
Send temperature to base.
No.
Is the button pressed D6?
Yes
Switch to mode 1.

Source Code

This is the source code for the base station controlling the mobot.

```
'{$STAMP BS2} 'STAMP directive ( a BS2)

sw1   var   byte  'variables for the various switches
sw2   var   byte
sw3   var   byte
sw4   var   byte
sw5   var   byte
sw6   var   byte
sw7   var   byte
sw8   var   byte

' THIS VARIABLEIS USED IN THE SERIN
‘_________________________________________________________________’

Msg   var   word  'variable used to store data received from serin
temp  VAR   Word  'used to store temperature value
dist  Var   byte
F     var   byte  'labels for moving in different directions, F = front
B     var   byte
R     var   byte
L     var   byte
ans   VAR   Byte  'bunch of flags , acts as flow control between the serins
FLAG1 VAR   BYTE
I     var   byte  'loop variable , used for the for statement
width VAR   nib   ' for passing data into the LCd
'_____________________________LCD vars_____________________________________

E           CON   11          ' LCD Enable pin  (1 = enabled)
RS          CON   10          ' Register Select (1 = char)

LCDddir     VAR   DirD        ‘ port direction
LCDout      VAR   OutD        ' 4-bit LCD data out
LCDin       VAR   InD         ' 4-bit LCD data in

ClrLCD      CON   $01         ' clear the LCD
DDRam       CON   $80         ' Display Data RAM control

char        VAR   Byte        ' character sent to LCD
index       VAR   Byte        ' loop counter

Flag        VAR   Byte

'________________________________________________________________________

' the variables used for the various subroutines and functions are
initalized, the LCD is warmed up here’

Initialize:
      temp = 0
      dist = 0
      F=0
      L=0
      B=0
      R=0
      ans = 0
      DirH = %11111100              ' setup pins for LCD
      GOSUB LCDinit                 ' initialize LCD for 4-bit mode
      char = ClrLCD                 ' clear the LCD
      GOSUB LCDcommand
      FOR index = 0 TO 6            ' create display in default mode
            LOOKUP index,["Welcome"],char
            GOSUB LCDwrite
      NEXT
      '______________________________________________________________

Main:

      BUTTON 1, 0, 1, 1, sw1, 1, butt1    ' 5 buttons to move fwd, bck,rt
and lft and uturn '

      BUTTON 2, 0, 1, 1, sw2, 1, butt2    ' when each button is pressed a
function is called that sets a flag'

      BUTTON 3, 0, 1, 1, sw3, 1, butt3
      BUTTON 4, 0, 1, 1, sw4, 1, butt4
      BUTTON 5, 0, 1, 1, sw5, 1, butt5
      BUTTON 7, 0, 1, 1, sw6, 1, butt6
      BUTTON 8, 0, 1, 1, sw7, 1, butt7
      BUTTON 9, 0, 1, 1, sw8, 1, butt8


      If(Flag1=7) THEN mode1
      If(Flag1=8) THEN mode2
      If(Flag=1) and (flag1 = 0) THEN Fwd 'here depeding on the flag sends
a serout

      If(Flag=2) and (flag1 = 0) THEN Bwd 'to the mobot to do something '
      If(Flag=3) and (flag1 = 0) THEN Left
      If(Flag=4) and (flag1 = 0) THEN Right
      If(Flag=5) and (flag1 = 0) THEN uturn


If(ans = 1) THEN Call_Serin 'serin coming in for the IR from the mobot '

If Msg = "4" THEN coll      ' MSg = 4 indicates collision

Goto main

'_________________________________________________________________________'

' this function is activated when msg = 4, used for the IR signal '

coll:
      F=0
      B=0
      L=0
      R=0

      char = DDRam                  ' show address at position 5
      GOSUB LCDcommand
      width = 15
      FOR index = 0 TO width        ' create display in default mode
            LOOKUP index,["Collision      "],char
            GOSUB LCDwrite
      NEXT
Goto Main
```

```
' it is the default handshake message between the mobot and the base at the
end of any action ‘

Call_Serin:
      ans = 0     ' ans is the flag that controls the function , function
is polled and called only when any
      Serin 6, 813, [Msg] ' buton is pressed
      debug msg
Goto main

' below are the various function for the robot based on the button pressed
from butt1 to butt8
'  has serouts and sets the flag up for the serin (IR) '


butt1:
      SEROUT 0, 16780, ["1"] ' Send the greeting.
      Flag=1
      ans = 1
Goto Main

butt2:
      SEROUT 0, 16780, ["2"] ' Send the greeting.
      Flag=2
      ans = 1
Goto Main

butt3:
      SEROUT 0, 16780, ["3"] ' Send the greeting.
      Flag=3
      ans = 1
Goto Main

butt4:
      SEROUT 0, 16780, ["4"] ' Send the greeting.
      Flag=4
      ans = 1
Goto Main

butt5:
      SEROUT 0, 16780, ["5"] ' Send the greeting.
      Flag=5
      ans = 1
Goto Main

butt6:
      SEROUT 0, 16780, ["6"] ' Send request to measure temperature.
      'Flag=6
      'ans = 1
      serin 6, 813, [Msg]
      goto disp_temp

butt7:
      SEROUT 0, 16780, ["7"] ' Send request to measure temperature.
      Flag1 = 7
goto main

butt8:
      SEROUT 0, 16780, ["8"] ' Send request to measure temperature.
      Flag1 = 8
goto main
```

```
' from here omn all messages are displayed depending on button pressed,
message is fwd, bk etc.. with units moved '

Fwd:
      char = DDRam                          ' show address at position 5
      GOSUB LCDcommand
      F=F+4
      B=0
      L=0
      R=0
      temp = F
      width = 4
      FOR index = (width - 1) TO 0        ' display digits left to right
            char = (temp DIG index) + 48  ' convert digit to ASCII
            GOSUB LCDwrite                ' put digit in display
      NEXT

      FOR index = (width-1) TO (width +10)
            LOOKUP index,["    cm Forward "],char
            GOSUB LCDwrite
      NEXT
      Flag = 0
Goto Main


Bwd:
  char = DDRam                              ' show address at position 5
  GOSUB LCDcommand
      F=0
      B=B + 4
      L=0
      R=0
      temp = B
      width = 4
      FOR index = (width - 1) TO 0        ' display digits left to right
            char = (temp DIG index) + 48  ' convert digit to ASCII
            GOSUB LCDwrite                ' put digit in display
      NEXT

      FOR index = (width-1) TO (width +10)
            LOOKUP index,["    cm Backward"],char
            GOSUB LCDwrite
      NEXT
      Flag = 0
Goto main

Left:
      char = DDRam                          ' show address at position 5
      GOSUB LCDcommand
      F=0
      B=0
      L=L+30
      R=0
      temp = L
      width = 4
      FOR index = (width - 1) TO 0        ' display digits left to right
            char = (temp DIG index) + 48  ' convert digit to ASCII
            GOSUB LCDwrite                ' put digit in display
      NEXT
      FOR index = (width-1) TO (width +10)
            LOOKUP index,["    Deg Left   "],char
            GOSUB LCDwrite
      NEXT
      Flag = 0

Goto Main

Right:
      char = DDRam                  ' show address at position 5
      GOSUB LCDcommand
      F=0
      B=0
      L=0
      R=R+30
      temp = R
      width = 4
      FOR index = (width - 1) TO 0        ' display digits left to right
            char = (temp DIG index) + 48  ' convert digit to ASCII
            GOSUB LCDwrite                ' put digit in display
      NEXT
      FOR index = (width-1) TO (width +10)
            LOOKUP index,["    Deg Right  "],char
            GOSUB LCDwrite
      NEXT
      Flag = 0
Goto Main

Mode1:
      char = DDRam                        ' show address at position 5
      GOSUB LCDcommand
      FOR index = 0 TO 15                 ' create display in default mode
            LOOKUP index,["    Mode1       "],char
            GOSUB LCDwrite
      NEXT
      Flag1 = 0
      serin 6, 813, [Msg]
      goto disp_temp

Goto Main


' the display message for the toggle between the two modes
Mode2:
      char = DDRam                        ' show address at position 5
      GOSUB LCDcommand
      FOR index = 0 TO 15                 ' create display in default mode
            LOOKUP index,["    Mode2       "],char
            GOSUB LCDwrite
      NEXT
      Flag1 = 0
Goto Main

uturn:
      char = ClrLCD                             ' clear the LCD
      GOSUB LCDcommand
      FOR index = 0 TO 5                  ' create display in default mode
            LOOKUP index,["U-TURN"],char
            GOSUB LCDwrite
      NEXT
      temp=0
      Flag = 0
Goto Main

' for displaying the temperature sent in from the mobot
disp_temp:
      char = ClrLCD
      GOSUB LCDcommand
      width = 4
      FOR index = (width - 1) TO 0              ' display digits left to
right
            char = (Msg DIG index) + 48         ' convert digit to ASCII
            GOSUB LCDwrite                ' put digit in display
      NEXT
      FOR index = (width-1) TO (width +10)            ' create display in
default mode
            LOOKUP index,["    Deg Celsius "],char
            GOSUB LCDwrite
      NEXT

Goto Main

' end of all messages , LCD function here on '

'_________________INITIALISING DISPLAY__________________________________'

LCDinit:
  PAUSE 500                           ' let the LCD settle
  LCDout= %0011                       ' 8-bit mode
  PULSOUT E,1
  PAUSE 5
  PULSOUT E,1
  PULSOUT E,1
  LCDout = %0010                      ' 4-bit mode
  PULSOUT E,1
  char = %00001100                    ' disp on, crsr off, blink off
  GOSUB LCDcommand
  char = %00000110                    ' inc crsr, no disp shift
  GOSUB LCDcommand
  RETURN
'________________
'LCDcommand:
  LOW RS                              ' enter command mode
LCDwrite:
  LCDout = char.HighNib                     ' output high nibble
  PULSOUT E,1                               ' strobe the Enable line
  LCDout = char.LowNib                      ' output low nibble
  PULSOUT E,1
  HIGH RS                             ' return to character mode
  RETURN
'_________________________________________________________________________'

'{$STAMP BS2} 'STAMP directive ( a BS2)
'------------------------------Variable declaration----------------------
---------------'
Get         var   byte        'Value gets in SERIN from Base-Station
pulse_count       var   byte  'used in FOR-NEXT loop
MoveType    var   nib(32)     'Array created for
                              'keeping track of the
                              'movements of robot and
                              'back-traversing of robot
                              ' in the same path
```

```
flag        var   byte        'Checks whether the Robot should
                              'move in increment or decrement
                              'of index
index       var   byte        'keeps track of the position in an
                              'array
'------------------------Variables  for temperature sensor------------------
----------'
DQ          CON   4                 ' DS1620.1 (data I/O)
Clock       CON   5                 ' DS1620.2
Reset       CON   6                 ' DS1620.3

RdTmp       CON   $AA               ' read temperature
WrHi        CON   $01               ' write TH (high temp)
WrLo        CON   $02               ' write TL (low temp)
RdHi        CON   $A1               ' read TH
RdLo        CON   $A2               ' read TL
StartC      CON   $EE               ' start conversion
StopC       CON   $22               ' stop conversion
WrCfg       CON   $0C               ' write config register
RdCfg       CON   $AC               ' read config register

tempIn      VAR   Word              ' raw temperature
sign  V     AR    tempIn.Bit8       ' 1 = negative temperature
tSign       VAR   Bit
tempC       VAR   byte              ' Celsius
'-----------------------------Variables  for IR --------------------------
------------------'
left_IR_det var   bit         ' Two bit variables for saving IR
right_IR_det var  bit         ' detector output values.3

'---------------------------------------- Initialization -----------------
----------------------'
index=1
flag=0
output 15                     ' signals to function as outputs for IR 1
output 3                      ' signals to function as outputs for IR 2
low 12                        ' pin used for Servo left wheel
low 13                        ' pin used for Servo right wheel
'------------------------Initial code for Temperature sensor---------------
----------'
HIGH Reset                    ' alert the DS1620
SHIFTOUT DQ, Clock, LSBFirst, [WrCfg, %10]      ' use with CPU, free-run
LOW Reset
PAUSE 10
HIGH Reset
SHIFTOUT DQ, Clock, LSBFirst, [StartC]    ' start conversions
LOW Reset
'--------------------------------Initialization Ends------------------'

'----------------------Main program starts-------------------------'
main:

Serin 0,16780,[get]
debug ? get
if (get=55) THEN main1
if (get=56) THEN main2
goto main
'---------------------------------Main Ends------------------------------'
'------------CODE :REMOTE CONTROL MODE2 STARTS-------------------'
main2:
```

```
if (get=49) THEN forwardNew         'Robot moves forward
if (get=50) THEN backwardNew        'Robot moves forward
if (get=51) THEN left_turnNew       'Robot moves forward
if (get=52) THEN right_turnNew      'Robot moves forward
if (get=53) THEN u_turnNew          'Robot moves forward
if (get=54) THEN Get_Temperature2   'Sensor DS1620 records
                                    'temperature
if (get=55) THEN main1              'changes the mode to mode1

' this is the primary comm lop, when any button is presed and the action
'executed, the signal is sent out in a loop so that IR serin of the base
receives it '

FOR pulse_count = 0 TO 50
      SEROUT 1,17197,["2"] ' Send the greeting.
NEXT

' this for the RF comm
SERIN 0,16780,[get]                 ' based on incoming signal flag is set
                                    'based on flag, mobot moves '
debug ? get
goto main2

'------------------------REMOTE CONTROL MODE ENDS----------------'

'------------------CODE: AUTONOMOUS MODE1 STARTS---------------'
main1:
if(flag=1) THEN GoBack
Gosub Forward
goto main1

Goback:
      index = index - 1
      if(index=0) THEN come
      if MoveType(index) =  1  then  forward2
      if MoveType(index) =  5  then  backward2
      if MoveType(index) =  2  then  right_turn2
      if MoveType(index) =  3  then  left_turn2
      'NOTE: left turn and right turn values are interchanged
      'Since the robot should behave diffrently while coming
      if MoveType(index) =  4 then  u_turn2
GOTO Main1

come:
      flag=0
      PAUSE 5000
      for pulse_count = 0 to 54
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
goto Main

go:
      flag=1
      PAUSE 2000
      for pulse_count = 0 to 54
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
      GOSUB Get_Temperature1
goto Main1

Forward:                                  ' If no detect, one forward
pulse.
      debug ? MoveType(index), ? index
      if (index=32) THEN go
      freqout 15, 1, 38500                ' Send freqout signal - left
IRLED.
      left_IR_det = in14                  ' Store IR detector output in
RAM.
      freqout 3, 1, 38500                 ' Repeat for the right IR pair.
      right_IR_det = in2
      if left_IR_det = 0 then right_turn  'takes LEFT on RIGHT-IR-DET
      if right_IR_det = 0 then Left_turn  'takes RIGHT on LEFT-IR-DET
      for pulse_count = 1 to 20
            pulsout 12, 500
            pulsout 13, 920
            pause 20
      next
      get = 0                             ' flag for communication
      MoveType(index) = 1
      index = index + 1
goto Main1

left_turn:                                ' Left turn routine.
      for pulse_count = 0 to 9
            pulsout 12, 500
            pulsout 13, 580
            pause 20
      next
      get = 0
      MoveType(index) = 2
      index = index + 1
goto Main1

right_turn:                               ' Right turn routine.
      for pulse_count = 0 to 9
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
      get = 0
      MoveType(index) = 3
      index = index + 1
goto Main1

u_turn:                                   ' U-turn routine.
      for pulse_count = 0 to 54
      pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
      get = 0
      MoveType(index) = 4
       index = index + 1
goto Main1

backward:                                 ' Used by each navigation routine

      for pulse_count = 1 to 10
```

```
            pulsout 12, 1000
            pulsout 13, 580
            pause 20
      next
      get = 0
      MoveType(index) = 5
index = index + 1
goto Main1

' *************************************************** '
forward2:                           ' If no detect, one forward pulse.

      for pulse_count = 1 to 20
            pulsout 12, 500
            pulsout 13, 920
            pause 20
      next
goto Main1
                                    ' Check again.
left_turn2:                         ' Left turn routine.
      for pulse_count = 0 to 9
            pulsout 12, 500
            pulsout 13, 580
            pause 20
      next
goto Main1

right_turn2:                        ' Right turn routine.
      for pulse_count = 0 to 9
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
goto Main1

u_turn2:                            ' U-turn routine.
      for pulse_count = 0 to 54
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
goto Main1

backward2:                          ' Used by each navigation routine .
      for pulse_count = 1 to 20
            pulsout 12, 1000
            pulsout 13, 580
            pause 20
      next
goto Main1

'------- Function to drive the DS 1620 temperature sensor-----------------'

Get_Temperature1:
      HIGH Reset                                ' alert the DS1620
      SHIFTOUT DQ, Clock, LSBFIRST, [RdTmp]     ' give command to read temp
      SHIFTIN DQ, Clock, LSBPRE, [tempIn\9]     ' read it in
      LOW Reset                                 ' release the DS1620
      tSign = sign                              ' save sign bit
      tempIn = tempIn / 2                       ' round to whole degrees
      IF (tSign = 0) THEN No_Neg1
      tempIn = tempIn | $FF00                 ' extend sign bits for
negative
No_Neg1:
      tempC = tempIn                          ' save Celsius value
      tempIn = tempIn */ $01CC                ' multiply by 1.8
      IF (tSign = 0) THEN No_Neg2             ' if negative, extend sign
      tempIn = tempIn | $FF00
No_Neg2:
      tempIn = tempIn + 32                    ' finish C -> F conversion
      'tempF = tempIn                         ' save Fahrenheit value
      FOR pulse_count = 0 TO 50
            SEROUT 1,17197,[tempC]      ' Send the temperature.
      NEXT
      Debug "TempC = ", dec tempC, cr
      get = 0
Goto Main1



'-----------------------------functions used in MAIN 2/ MODE 2-------------
---------------------------'

' when collison occurs this spl flag is set and a different message is
passed to the IR
'on the base from here on the functions to the servo for movement '

forwardNew:                             ' If no detect, one forward
pulse.
      for pulse_count = 1 to 20
      freqout 15, 1, 38500              ' Send freqout signal - left
IRLED.
      left_IR_det = in14                ' Store IR detector output in
RAM.
      freqout 3, 1, 38500               ' Repeat for the right IR pair.
      right_IR_det = in2

      if left_IR_det = 0 then collision
      if right_IR_det = 0 then collision
      pulsout 12, 500
      pulsout 13, 920
      pause 20
      next
      get = 0                           ' flag for comm
goto main2                              ' Check again.

left_turnNew:                           ' Left turn routine.
      for pulse_count = 0 to 9
            pulsout 12, 500
            pulsout 13, 580
            pause 20
      next
get = 0
goto main2

right_turnNew:                          ' Right turn routine.
      for pulse_count = 0 to 9
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
get = 0
```

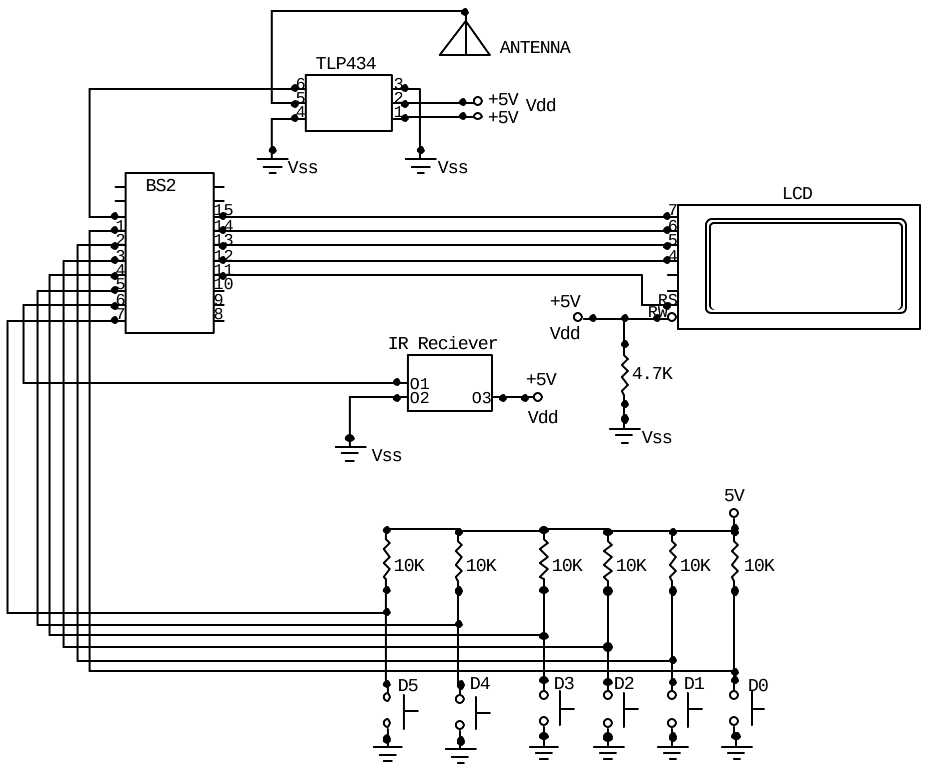
```
goto main2

u_turnNew:                                  ' U-turn routine.
      for pulse_count = 0 to 54
            pulsout 12, 1000
            pulsout 13, 920
            pause 20
      next
get = 0
goto main2

backwardNew:                                ' Used by each navigation routine
.
      for pulse_count = 1 to 20
            pulsout 12, 1000
            pulsout 13, 580
            pause 20
      next

get = 0
goto main2

' this backward is called when a collision is detected, usually for moving
back the previous function '

backwardNew1:                               ' Used by each navigation routine
.
      for pulse_count = 1 to 10
            pulsout 12, 1000
            pulsout 13, 605
            pause 20
      next
get = 0
GOTO Main2

collision:
      FOR Pulse_count = 0 TO 50
            SEROUT 1,17197,["4"]            ' Send the greeting.
      NEXT
      Gosub BackwardNew1
Goto Main2
'Return

' Function to drive the DS 1620 temperature sensor.

Get_Temperature2:
      HIGH Reset                                  ' alert the DS1620
      SHIFTOUT DQ, Clock, LSBFIRST, [RdTmp]       ' give command to read temp
      SHIFTIN DQ, Clock, LSBPRE, [tempIn\9]       ' read it in
      LOW Reset                                   ' release the DS1620
      tSign = sign                                ' save sign bit
      tempIn = tempIn / 2                         ' round to whole degrees
      IF (tSign = 0) THEN No_Neg11
      tempIn = tempIn | $FF00                     ' extend sign bits for
negative
No_Neg11:
      tempC = tempIn                              ' save Celsius value
      tempIn = tempIn */ $01CC                    ' multiply by 1.8
      IF (tSign = 0) THEN No_Neg21                ' if negative, extend sign
bits
      tempIn = tempIn | $FF00
```

```
No_Neg21:
      tempIn = tempIn + 32                          ' finish C -> F conversion
      'tempF = tempIn                               ' save Fahrenheit value
      FOR pulse_count = 0 TO 50
            SEROUT 1,17197,[tempC]                  ' Send the temperature.
      NEXT
      Debug "TempC = ", dec tempC, cr
      get = 0
Goto Main2
```

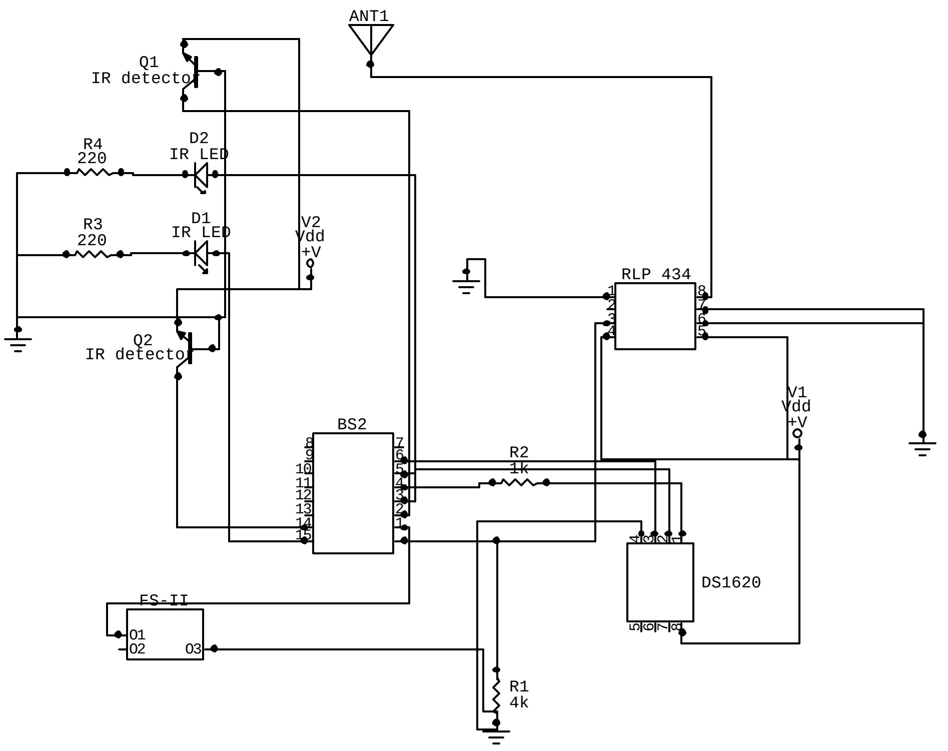
‘ ------------------------------------ End of all Programs --‘

Circuit Diagrams

Base Station Circuit



Mobot Circuit


ANT1
Q1
IR detector
R4
220
D2
IR LED
R3
220
D1
IR LED
V2
Vdd
+V
Q2
IR detector
RLP 434
V1
Vdd
+V
BS2
R2
1k
DS1620
FS-II
01
02
03
R1
4k