Mechatronics
MAE 476/576

ROBOT HUNTER

ROBOT HUNTER

Submitted by:
Rajani Boddu
Craig Cole
Sai Krishna Prasad Gavernini
Preeti Sadanand Joshi
Vikranth Bejja Rao

Index

Acknowledgement

We are very thankful to Dr. Krovi for inspiring us for doing a challenging project. The kits provided by him were of great help in the project. We are also thankful to Mr. Chin-Pei Tang, our Teaching Assistant, who was always available to help us throughout the project. We extend our regards to Mr. Craig Cole for investing in the Parallax Sumo Bot for the project.

Last but not the least, we thanks the other teams for their cooperation and for helping us by providing the components required in this project.

1. Abstract

The aim of this project was to develop a system with distributed sensing and control. The system comprises of a base station (BS2 Board of education), a mobile station (BOE-BOT Kit) which are coordinated with wireless communication. The report describes the features of our system “Robot Hunter” with the need, functions and implementation of various components which were integrated to form the system. The various components used, apart from the Basic Stamp 2 boards were ultrasonic sensors, parallax servos, IR LEDS, IR receivers, Whiskers, TWS-434 transmitter and RWS-434 receivers and other common peripherals. The report also explains the principles implemented to use the above components to form a useful system. Some of the principles were real-time data logging, RF communication, servo calibration, ultrasonic ranging and focusing, IR sensing, Alpha-Beta estimation algorithm, tactile sensing using whiskers. A copy of PBASIC code is also attached in order to enhance the understanding the system.

2. Problem Statement

The objective of the project was to develop and implement a distributed sensing and control framework for a system, mobile robot. The system was required in the operate in the way defined: a distributed sensing mode, where either base station or mobile robot serves as a source of acquiring data from the outside world; this information being communicated to the other system, processed and result in to an action performing agent. The main challenge of this project was the working of multiple processors in tandem and coordinate actions required for an intelligent distributed sensing system. The project was open ended with freedom of constructing any system, which meets the basic requirements for a distributed system. These requirements were use of a base station and mobile robot, communication protocol (synchronous vs. asynchronous) and wired interfaces such as 7-segement displays, LCD, buttons etc.

3. Motivation

Since the beginning of our semester we have been thinking of building a system which would be challenging and would require an extra mile to be covered. The underlying concept of our project was based on guided missile interception with the target. Keeping this at the back of our mind, we decided to develop a "***Robot Hunter Interception System***" **(RHIS)** which would track the presence of an intruder in our environment and capture it when given its position in space.

Our system, **RHIS** can be divided in to two major subsystems:

Base Station: Tracks the presence on the intruder

Rover: Moves and intercepts the target.

The following sections would describe the two systems in detail.

4.List of Components and Cost Estimate

S.no.	Name of the Component	Quantity Used	Price
1	Basic Stamp II Kit	1	$149
2	BOE BOT KIT	1	$229
3	Devantech SRF-04 Ultrasonic Sensor	1	$37.75
4	Parallax Standard Servo #900-00005	1	included in the Kit1
5	Rentron TWS-434A	1	included in the Kit2
6	RWS-434	1	included in the Kit2
7	Hitachi-compatible 2x16 Parallel LCD Display	1	included in the Kit1
8	IR LEDS	2	included in the Kit2
9	IR Receivers	2	included in the Kit2
10	Whisker	1	included in the Kit2
11	Batteries 1.5V , 9 V		

5. Base Station

5.1 Overview

The purpose of the base station is to locate stationary and moving objects in the arena, and communicate coordinates to the Hunter such that the Hunter intercepts it. To locate objects in the arena, a Devantech SRF-04 ultrasonic sensor is mounted to a servo that sweeps the sensor in an arc.

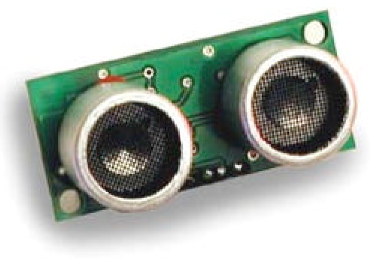
5.2 Major Components

There are three major subsystems that make up the base: ultrasonic sensor, servo and RF transmitter. In addition, a LCD Display is used to communicate information to the user. How each is used is described below.

Parallax Standard Servo #900-00005

Data Sheet:
http://www.parallax.com/Downloads/Documentation/Servo_Standard_Manual.pdf

The servo is used to point the ultrasonic sensor such that a target's angle and range can be determined. It is simple to use, but requires a significant amount of time for it to move to any position specified by the microcontroller. It has a range of 180 degrees, but is only used for 90 degrees of arc to prevent a single sweep of the arc from being impractically long.

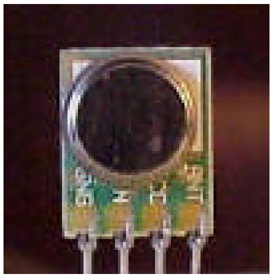


Devantech SRF-04 Ultrasonic Sensor:

Data Sheet:
http://www.robot-electronics.co.uk/shop/Ultrasonic_Ranger_SRF041999.htm

The ultrasonic sensor is used as the “long-range” sensor of the project, as it has a rated range of 3 meters. Some experimentation is required to determine what size object is necessary to be found by the sensor at long range, and one must be aware that it detects anything within a particular cone of sensitivity.

Once a start is pulsed to the ultrasonic sensor, it emits a pulse. Using the RCTIME command, with the use of 2-microsecond increments between emissions of the signal and recording the echo. It is fortunate that the BASIC Stamp operates with 2 microsecond increments, since it is equal to the number of microseconds taken for the signal to go one way. Sound travels takes 29 microseconds to travel one centimeter, so to convert the result to centimeters, the one-way travel time needs to be divided by 29. To convert to inches, the one-way travel time should be divided by 72. The scale is set at the top of the source code as a constant.



Rentron TWS-434A

Data Sheet:
http://www.rentron.com/remote_control/TWS-434.htm

This device enabled wireless communication from the base to the Hunter. Through it, the coordinates the Hunter is to move to are communicated.

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this pictu

Hitachi-compatible 2x16 Parallel LCD Display

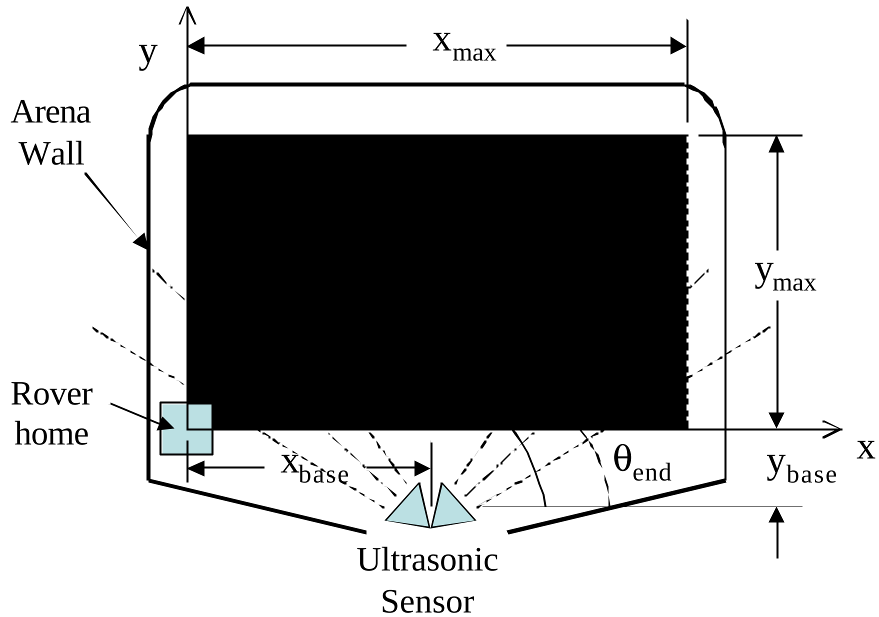
Data Sheet:
http://www.parallax.com/Downloads/Documentation/audiovis/Parallel_LCD_Manual.pdf

The LCD is used to communicate to the user the range and angle to the target, Hunter coordinates and other useful information, so that the unit can be operated without a terminal.

5.3 Theory of Operation

5.3.1 Arena

An overhead view of the arena is shown below:



The Hunter has a movement limit of 256 cm by 256 cm, so the arena was built so as to take advantage of as much movement as possible. The base station is placed somewhere near between the left and right sides of the arena, ideally in the middle, and set back somewhat. It is set back so as to minimize the size of the bottom corners that are not seen by sensor.

For our demonstration, the area was 220 cm wide and 230 cm long. The base was placed such that the center of the ultrasonic center was positioned 110 cm to the right of the Hunter home position and set back 60 cm from the Hunter home x-axis. These variables need to be entered into the base station microcontroller source code for the base station to function correctly.

The walls for our arena were constructed of 1/2" thick foam thermal insulation that was 18 inches tall. It is important that the walls be high enough to prevent the ultrasonic sensor from detecting objects behind it, though another constant can be set to limit the maximum range as reported by the sensor. In addition, the walls need to be as smooth as possible with rounded corners and any seams in the wall should be taped. This helps the base station to measure the range to the boundary as many times as possible.

5.3.2 Initialization

The first thing the base does upon power up is beep and give a short introductory message on the LCD display. It then moves the servo to its 45 degree position, which for our project should face straight forward into the arena. It allows the user a moment to make sure the sensor is correctly aimed.

After aiming the sensor, the user should make sure the arena is clear of potential targets. The user then pushes button D0, called the “START” key, to allow the base station to perform a background scan.

5.3.3 Background Scan

The way the base station knows a target has entered the arena is that it stores the arena boundaries into the microcontroller’s EEPROM and then scanning for any significant reduction in range. The scanning is done over a 90 degree arc in 3 degree increments. The total sweep angle and angle increment can easily changed by changing a few constants in the code, but this seemed to be a good medium between target range and angle accuracy and scan rate. There is a short subroutine in the program used to calculate the length of the pulse required to be sent to the servo based on the angle desired. Zero degrees is a line from the ultrasonic sensor moving parallel to the Hunter x-axis, and in the same direction.

Due to unavoidable noise in sensor measurements, it is likely that taking only one measurement will not be a reliable way to determine the boundaries of the arena. If the measurement is incorrect, when the base station scans for a target, it will incorrectly determine a target object position, when the arena background was incorrectly measured. Taking the average of several measurements is one solution, using the alpha-beta filter (already coded) for tracking the object, improved the estimate of the arena boundaries.

Five measurements are taken at every angle of arc, and are processed by the alpha-beta filter before storing the result in the microcontroller's EEPROM. Even if one measurement has a large error associated with it, the alpha-beta filter does a very good job of keeping its effects on the estimate to a minimum. To keep maximum resolution, the range time, not the distance, is stored in EEPROM.

The range measurements can be viewed using Stamp-Plot Lite, turning Stamp-Plot Lite into the Stamp Plot Sonar-Scope®. Though not particularly useful, it is interesting to see how the ultrasonic sensor "*sees*" the world.

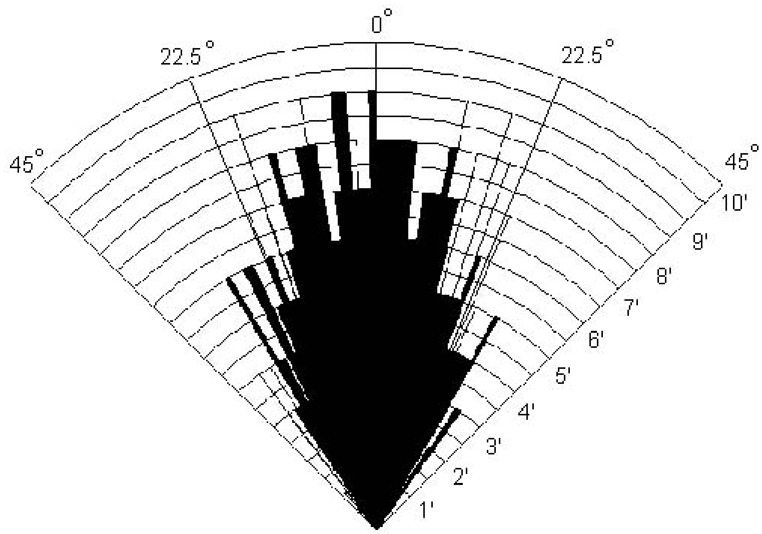
Once the base station has finished scanning the background ,it beeps and awaits the user to press the START button to begin scanning.

5.3.4 Scanning

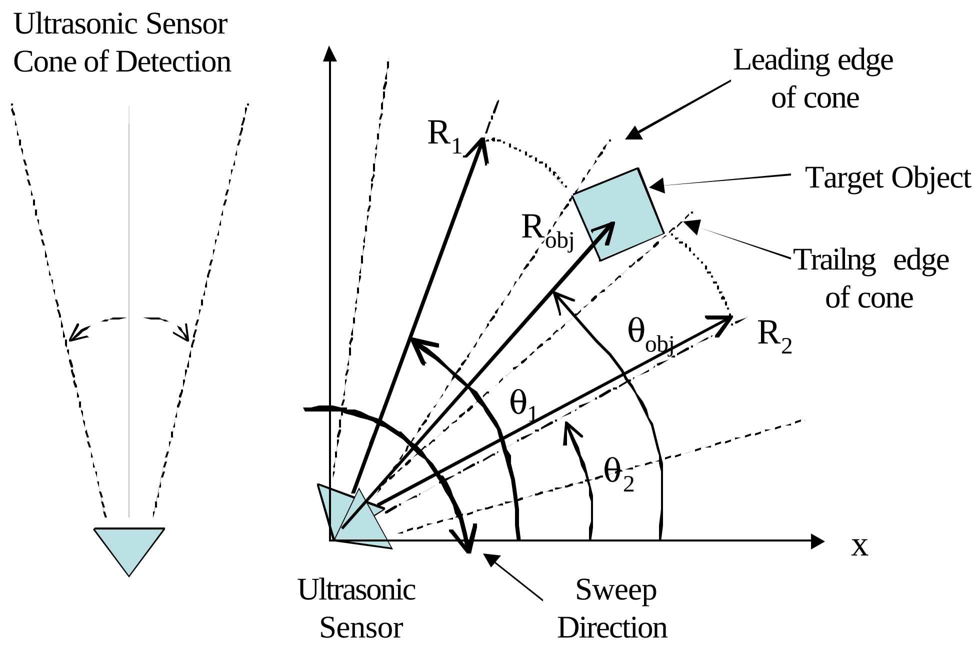
The main loop for scanning calls a subroutine to sweep the area with the ultrasonic sensor. This subroutine moves the head over a 90 degree arc, at three degree increments, as done during the background scan, and looks to see if it has changed significantly. The amount of difference allowed is set at the top of the code as a constant, and is currently set to 10 cm. If this value is set too low, the base station will mistakenly think it sees a target. If this value is too high, a target near the wall will not be seen.

If no target is seen at the end of the scan, it is indicated as such on the LCD display and the scan repeats.

It should be understood that the ultrasonic sensor is sensitive to not only things directly in front of it, but an object that lies within a 22.5 degree cone that emanates from the sensor as shown here from the sensor data sheets.



As the sensor sweeps clock-wise, when a target is found, it is likely that the object is being detected at the right leading edge of the cone, which means the object can be as much as 22.5 degrees below the current angle of the sensor. As a result, when a target it detected, the range and direction are stored, and the scan continued. When the object is no longer seen, it is likely because the object has left the trailing edge of the cone. The actual range and angle of the target is computed as the average of the range and target when first seen and when last seen by the sensor. This is detailed in the graphic below:



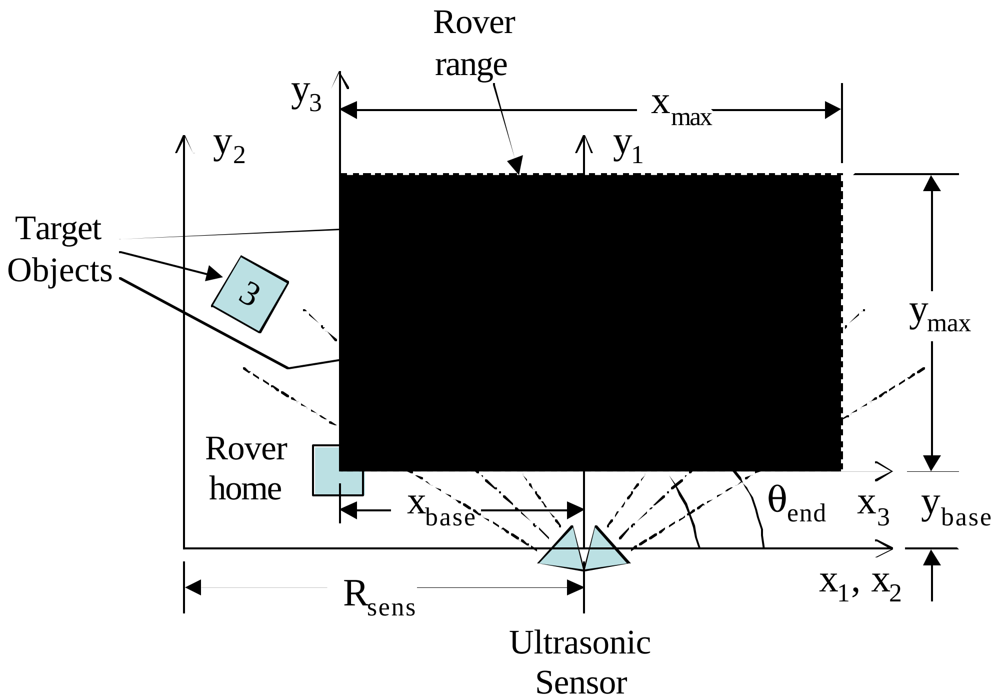
$$\theta_{obj} = (\theta_1 + \theta_2) / 2$$
$$R_{obj} = (R_1 + R_2) / 2$$

The base station emits a high-pitched beep to indicate a target has been spotted, and then emits a low-pitched beep to indicate when it no longer sees it. It is a useful bit of feedback, as it becomes easy to see when the sensor has picked up the intended target or has found something else of apparent interest. The calculated range and angle are output on the LCD display. The range is also displayed on the Stamp-Plot Sonar-Scope®, though when a target it being tracked, the amount of sweep viewed drops.

After a target's range and angle have been calculated, the subroutine changes the angle at which it starts its sweep to reduce the time its takes to locate the object on the next sweep. Control then returns to the main scan loop.

5.3.5 Coordinate Conversions

To obtain the values of coordinates (x, y) from the range, in microseconds, and angle from the alpha-beta filter involved a lot of calculations for coordinate transformations. A graph depicting the situation is shown below:



$$\theta_{sweep} = \theta_{start} - \theta_{end}$$

The main scan then converts the range and angle to (x,y) coordinates relative to the base. The angle had to be converted to binary radians so that the microcontroller's trigonometric functions could be used. The outputs of the sine and cosine functions are also odd: it is a word with a number ranging from 0 to 127, indicating 0 to 1. It is odd, because if the range had been 0 to 255, the output of the cosine and sine functions could be "multiply-middled" to the range to get the x and y components. Rather than multiply the range by the cosine or sine (which would have resulted in overflows because the range numbers are large anyway) and then divide by 127, the output of the sine and cosine were left-shifted one bit then the multiply-middle operation performed.

The alpha-beta filter will not work with negative numbers, so the coordinates are shifted to the coordinate system indicated by the x_2, y_2 axis. The origin is the same distance i.e. The maximum range of the sensor as set at the top of the program. This prevents any negative coordinates. These are the coordinates stored in the EEPROM.

Once stored, the scan subroutine is called again and the process repeats until five coordinates are stored. The number of measurements can be changed if desired by changing a constant at the top of the program. Five measurements seems to be just enough for the alpha-beta filter to sufficiently converge on a position. However, about ten measurements are needed for the filter to converge on velocity, which is discussed further in the section covering the alpha-beta filter.

5.4 Alpha-Beta Estimation

The Alpha-Beta filter is an estimation method used to determine the position and velocity of a moving (or stationary) target when faced with noisy, error-laden data. It is commonly used to track aircraft, and has many uses in the outside world. The fact that it can be worked into the BASIC Stamp shows that it is not computationally expensive and is reasonably fast.

For it to work correctly, however, it requires enough data points to converge on a solution. In our case, five measurements were enough for the routine to settle on a position, but not velocity. Tests showed that at least ten measurements would be needed, which was inconvenient at best. The best the scan routine can accomplish is about one scan every two seconds, mostly because the servo is slow and requires a relatively long time to settle at a position. This means for the alpha-beta filter to get enough measurements to get a good velocity solution, the object would have to be tracked for about 20 seconds. Not only would the object have to remain within the arena over 20 seconds, there would be lead distance left-over for the Hunter to intercept it. It would require an exceptionally slow moving object to meet those constraints.

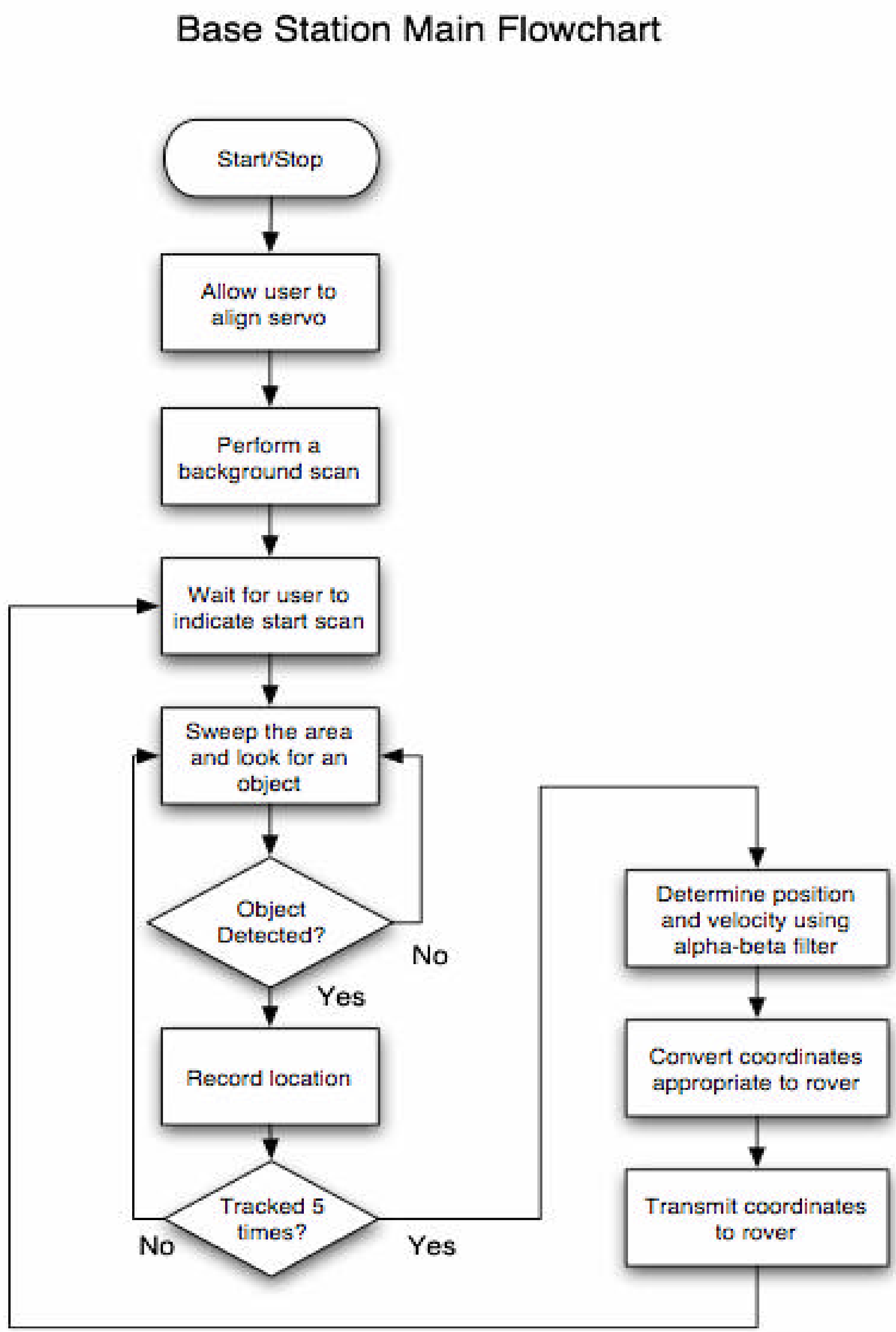
So, for this project, we decided to use stationary targets. To prevent the false velocities from affecting the results, the lead time given to the Hunter was zero seconds. The Hunter is essentially just going to the targets position, and not trying to lead it. If the servos controlling the ultrasonic sensor were replaced with a stepper motor, the sweep rate could be increased, the number of measurements increased and lead time reset to an appropriate length of time. The lead time is set at the top of the program as a constant.

5.5 Transmission to the Hunter

Once the alpha-beta filter computes coordinates, they are still relative to the x_2, y_2 axis and need to be shifted to the rover coordinate axis: x_3, y_3 and converted to centimeters. If the determined coordinates fall outside the range of the rover, a failure message is indicated on the LCD and no coordinates are transmitted. The base station waits for the user to press START and the base starts scanning again. Note that the background scan does not need to be run again, as it is still stored in EEPROM.

If the coordinates are okay, they are transmitted via RF to the rover and the coordinates displayed on the LCD display. The base then waits for the user to press START and the scanning process begins again.

5.6 Flow Chart



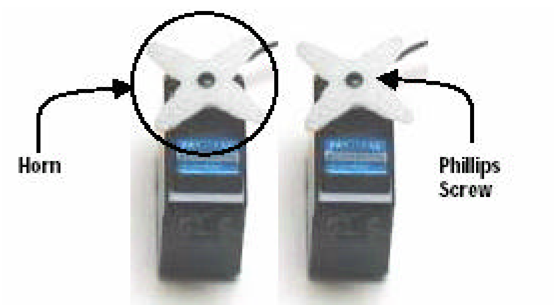
6. Rover Station (Moving BOT)

6.1 Overview

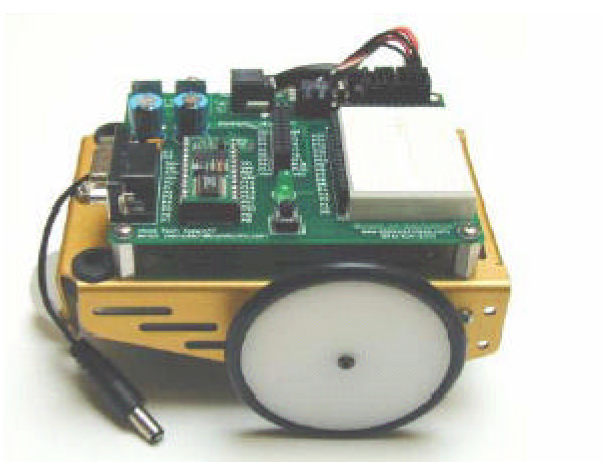
The Rover after receiving the position of the target moves thru space to the specified location. Once it reaches the position does a local scan and intercepts the target. The purpose of local scan is to confirm the target's presence. It is also possible that the coordinates transmitted by Base may not be accurate, so this enables the Rover to determine the correct location of the target.

6.2 Major Components

Parallax Pre-modified Servos

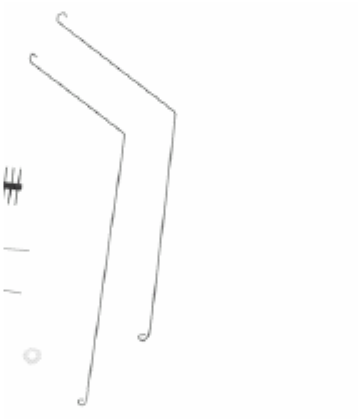


Assembled BOE BOT



(b)

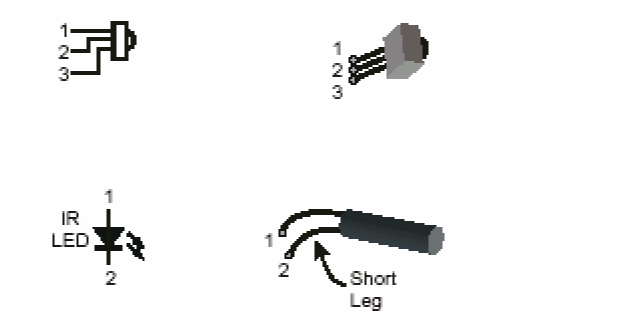
Nylon Whiskers Size #4



IR Receiver and IR LE

Data Sheet

http://www.rentron.com/Files/rf.pdf



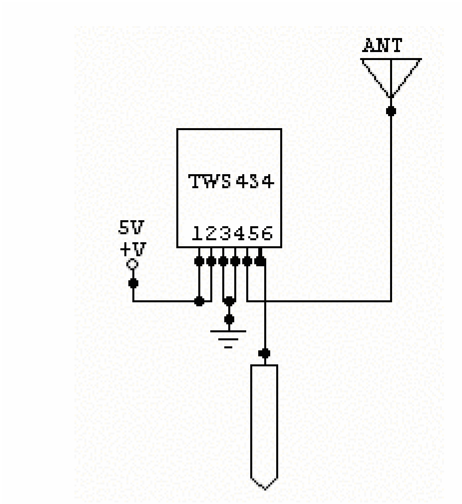
6.3 Theory of operation

Rover receives data (x and y coordinate positions of the intruding object) from Base station through antenna by RF Receiver. The receiver then decodes the received information. We have used the "go" for coding in our transmission of data. Once the data is decoded, rover is made to move in x and y directions individually by setting the motor pulses for the x, y coordinates. Calibration is done for the number of pulses and the distance traveled by rover to get the constants. Once the rover reaches to the target position it starts sweep around continuously to find the object in a span of 120 degrees. IR sensors which work on the principle of reflection of infrared light send from a source when obstructed by some target, helps in detecting the intruder by the rover. We incorporated the feature of detecting the object further by IR sensor to support for any errors in measurement of coordinates by the base station sensor, signal transmission errors, or errors in the rotation of rover servos or for any error in the initial setting (position and direction) of the Rover. The range of IR sensor is 30 cm which is good enough to track object in small range which is our requirement.

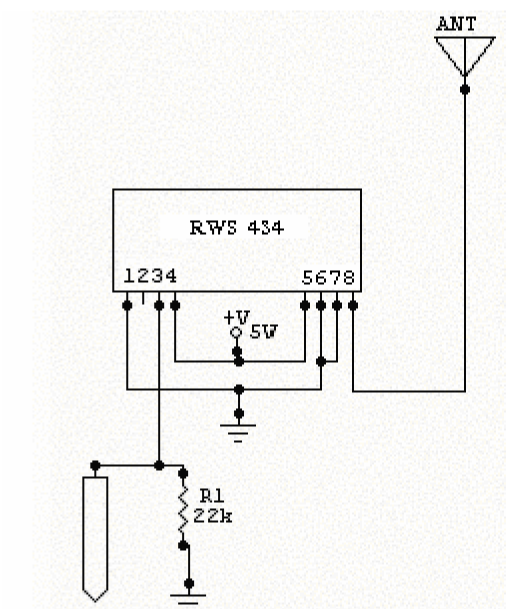
Once it detects any object it starts moving towards that direction. A tactile sensor, whisker, which is a mechanical extension to ground in the form of a wire, is hanged in front of the rover. When the rover approaches the object, the tactile sensor is the one that touches with the object first being it is in the front. This causes the sensors lean wire to spring/bend back making one of the ports, deliberately chosen, being grounded with it due to its backward movement. Thus turning of that port from high to low indicates the tracking of the object by the rover. Once tracking is done the further moving of rover is stopped.

7. Circuit Diagrams and I/O Pin Chart

7.1 Interfacing TWS-434

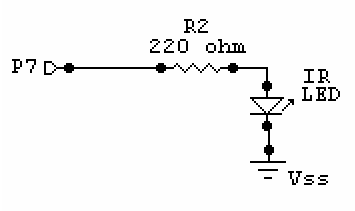


7.2 Interfacing RWS 434 (RF Communication)

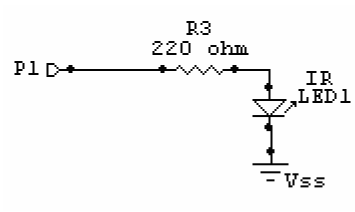


7.3 Interfacing IR LEDS

IR LED 1 :

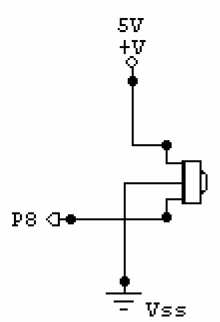


IR LED 2: (note port no is different)

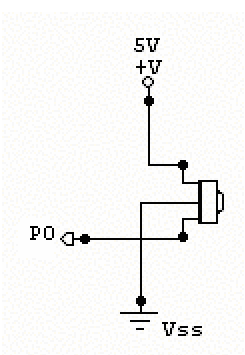


7.4 Interfacing IR Receivers

IR Receiver 1 :

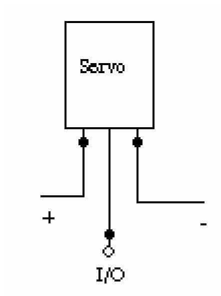


IR Receiver 2 (Note the port no is different):

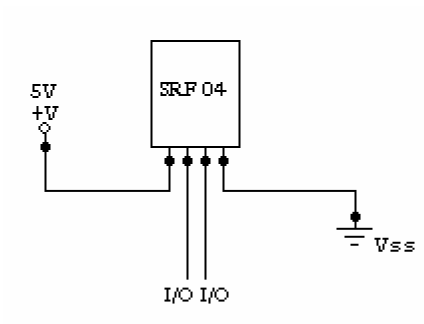


7.5 Interfacing Servos

Servo: (on base station)



7.6 Ultrasonic Sensor:



7.6 Base Station I/O Assignments

Address	I/O	Description
0	Output	Servo Output
1	Input	Ultrasonic Sensor pulse input
2	Input	Ultrasonic Sensor Echo input
3	Output	RF Transmitter
4	Output	Piezo Beeper
5	Output	LED Indicating Tracking
8	Output	LCD LSB
9	Output	LCD
A	Output	LCD
B	Output	LCD MSB
C	Output	LCD E
D	Output	LCD R/S

7.7 Rover (Boe-Bot) I/O Assignments

Address	I/O	Description
0	Input	IR Receiver
1	Output	IR LED
2	Output	Piezo Beeper
3	Input	RF Receiver
4	Input	Whisker
7	Output	IR LED
8	Input	IR Receiver

8. Problems during implementation

The main problem we faced was the calibration of the servos of the rover which was dependent on the power supply. When the battery got discharged, power used to go low, the servos would have to be provided with a different pulse to move a specified distance. So, we always used fresh batteries in our project. The servos were also calibrated according to the surface on which it was moving.

Basic Stamp 2 does not support trigonometric functions and signed division with floating point numbers. This made the writing of code tedious and complicated. Limited amount of memory was a problem again. We had to optimally utilize the 2K memory of the BS2.

One more problem we faced was loose circuit. The IR sensors mounted on the breadboard on the over did not always make a good contact on the breadboard. Before running the project, we made sure that the IR sensors were well made contact on the breadboard.

RF receiver was mounted on the rover. It requires 4.5V to 5.5V while all the other components on the rover require 6V for proper operation. We have used four 1.5V batteries for the rover. For the RF transmitter, we tapped supply form 3 batteries to provide 4.5V to transmitter.

9. Contribution

Rajani Boddu: Concept of the project
Development, implementation and calibration RF communication.
Development and Calibration of Rover
Testing and debugging of the Project
Project Report.

Craig Cole : Concept of the project
Implementation of Ultrasonic sensor on Base station, **Alpha beta estimation algorithm**
Calibration of Base station in determining position
Testing and debugging of the Project.
Project Report.

Sai Gavirneni: Testing and debugging of the Project
Project Report.

Preeti Joshi: Concept of the project
Development, implementation and calibration of RF communication,
Implementation of Sensors on Rover
Testing and debugging of the Project.
Project Report.

Vikranth Rao : Concept of the project
Development of Rover Station
Calibration of Servos, complete **motion control of Rover**
Interfacing of Whiskers
Testing and debugging of the Project.
Project Report.

10. Conclusions

We have successfully implemented a project which includes distributed computing between two basic stamps - one on the base station and one on the Rover. RF transmission and reception is implemented as means of communication between the two processors. This project is a good example of an intelligent system using hardware and software. This project gave us an insight in optimally utilizing the memory and I/O ports. It also gave us an opportunity to work in a group with coordination and cooperation.

11. Code

11.1 Base Station

```
'{$STAMP BS2}
'===============================================================================
=====
'
' Final Propject: Base Station                    7 May 2003
'                                              MAE 576 Mechatronics
' LAB GROUP C
'                                              FINAL VERSION!
' Rajani Boddu
' Craig Cole
' Sai Krishna Prasad Gavernini
' Preeti Sadanand Joshi
' Vikranth Bejja Rao
'
'===============================================================================
=====

'-------------------------------------------------------------------------------
-----
'Base Control Parameters

minDeg          CON   45    'Start of Scan (Note! 90 degrees is parallel to y-
axis)
maxDeg          CON   135   'End of scan   (Same...)
angleStep   CON   3     'Degrees of sweep per step
maxRange    CON   300   'Maximum allowable range (cm)
maxErr          CON   15    'Maximum allowable error wo/targeting (cm)
sweep       CON   3     'Sweep starts sweep*angleStep before target
rOffset         CON   -10   'Range offset (cm)
aOffset         CON   0     'Angle offset
yretreat    CON   3     'Distance to allow rover to track

baseX       CON   110   'X-distance from origin to base (cm)
baseY       CON   30    'Y-distance from origin to base (cm)
roverXmax   CON   220   'Maximum x range of rover
roverYmax   CON   230   'Maximum y range of rover
scale       CON   29    'Unit Scale (29=cm, 72=in)

trackReq    CON   5     'Number of points needed for alpha-beta
leadtime    CON   0           'Number of seconds lead required for rover
dt          CON   $0200 'Time between measurements (sec)
alpha       CON   $00a0 'Alpha = 0.5
betaOverDT  CON   $0020 'betaOverDT = beta/dt where Beta = 0.172 & dt=2 sec

yes         CON   1
no          CON   0

'-------------------------------------------------------------------------------
-----
'Stamp IO Ports
```

```
servo       CON   0     'Servo pin
init        CON   1     'Ultrasonic Sensor Pulse pin
echo        CON   2     'Ultrasonic Sensor Echo pin
xmitter           CON   3     'RF trasmitter
piezo       CON   4     'Piezo speaker
ontargetLED CON   6     'LED indicating ontarget
cnote       CON   2091  'c note
dnote       CON   2348  'd note
enote       CON   2636  'e note
baud        CON   17197 'Baud settings


'-----------------------------------------------------------------------------------
-----
'Variables

i           VAR   byte  'Workhorse index variable
memAddress  VAR   word  'EEPROM memory pointer for Alpha-Beta
pulse       VAR   word  'Pulse to control servo
deg         VAR   word  'Servo Angle
degStart    VAR   byte  'Start of sweep
deg1        VAR   word  'Angle at beginning of target

R           VAR   word  'Range to target
R0          VAR   word  'Background range
R1          VAR   word  'Range at beginning of target
ontarget    VAR   bit   'Indicates if currently in middle of target
lastR       VAR   word  'Used to track target closest to base
targetX           VAR   word  'Target x-coordinate (cm)
targetY           VAR   word  'Target y-coordinate (cm)
trackNum    VAR   nib   'Number of times target tracked

StartBtn    VAR   In5   'Button to start scanning again

roverx            VAR   byte  'Final x-coord for rover
rovery            VAR   byte  'Final y-coord for rover

memaddr2    VAR   targetx     'Used for BG Scan

'Alpha-Beta section reuses as many variables as possible

rm1         VAR   R     'Current measurement
rp1         VAR   R0    'Predicted measurement
rs0         VAR   lastR 'Previous
vp1         VAR   pulse 'Predicted velocity
vs0         VAR   targety     'Previous
rs1         VAR   R1    'Smoothed position
vs1         VAR   deg1  'Smoothed velocity

'-----------------------------------------------------------------------------------
-----
'LCD Related Variables & Constants

E           CON   12    ' LCD Enable pin  (1 = enabled)
RS          CON   13    ' Register Select (1 = char)
LCDout            VAR   OutC  ' 4-bit LCD data
```

```
ClrLCD      CON   $01         ' clear the LCD
CrsrHm      CON   $02   ' move cursor to home position
CrsrLf      CON   $10   ' move cursor left
CrsrRt      CON   $14   ' move cursor right
DispLf      CON   $18   ' shift displayed chars left
DispRt      CON   $1C   ' shift displayed chars right
NxtLn       CON   $C0   ' Move cursor to start of 2nd line
DDRam       CON   $80   ' Display Data RAM control

char        VAR   pulse.lowbyte     ' character sent to LCD
LCDPtr            VAR   R0    ' Points to LCD te is to display
index       VAR   I     ' Index for loops in LCD subroutine
number            VAR   R1    ' Put numbers to be displayed/entered here
zeroBlank   VAR   bit   ' if true, zeros still okay to blank

'----------------------------------------------------------------------------------
-----
'EEPROM Data

            DATA  @0, 0 (120) 'Clear out space for Scan Data

IntroMsg    DATA  ClrLCD,CrsrHm
            DATA  " Sonar Station",NxtLn
            DATA  "Team 3     2003",0

AlignMsg    DATA  ClrLCD,CrsrHm
            DATA  "  Align Base,",NxtLn
            DATA  " & press START.",0

StartMsg    DATA  ClrLCD,CrsrHm
            DATA  "  Press START",NxtLn
            DATA  "    to scan:",0

BGMsg       DATA  ClrLCD,CrsrHm
            DATA  "Background Scan",0

ScanMSG           DATA  ClrLCD,CrsrHm
            DATA  "Scanning...",0

NoTargetMsg DATA  ClrLCD,CrsrHm
            DATA  "No target.",0

TrackMsg    DATA  ClrLCD,CrsrHm
            DATA  "Track No.",0

AngleMsg    DATA  NxtLn
            DATA  "Ang=",0

RangeMsg    DATA  "  Rng=",0

RoverXMsg   DATA  ClrLCD,CrsrHm
            DATA  "Go ( ",0

RoverYMsg   DATA  ",",0

RoverEndMsg DATA  " )",NxtLn
            DATA  "Press START...",0
```

```
FailMsg            DATA  ClrLCD,CrsrHm
            DATA  "Can't Intercept",NxtLn
            DATA  "Press START...",0

'===================================================================================
'
'Initializaton
'
'===================================================================================

  DIRS = %1111111111011011
'         FEDCBA9876543210

  GOSUB LCD_Init

  LCDptr = IntroMsg
  GOSUB Write_LCD
  FREQOUT piezo, 1000, dnote
  PAUSE 2000

Start:

'Give a moment for base to be aligned Vertically

  LCDptr = AlignMsg
  GOSUB Write_LCD

  deg = 90
  GOSUB Calc_Pulse
Loop2:
    PULSOUT servo, pulse
    PAUSE 10
  if StartBtn = 1 then Loop2:

'Reset to zero degrees

  deg = maxDeg
  GOSUB Calc_Pulse
  FOR i = 0 to 50
    PULSOUT servo, pulse
    PAUSE 10
  NEXT

'===================================================================================
'
'Scan the background and record in EEPROM
'
'===================================================================================

  GOSUB Setup_Plotter         'Setup Stamp Plot Lite for "Radar Scope"

  LCDptr = BGMsg
```

```
  GOSUB Write_LCD

  memAddr2 = trackReq*4 'Right after x,y coords in EEPROM
  memAddress = 0  'Store readings for alpha-beta

BGLoop:
  GOSUB Calc_Pulse
  FOR i = 0 to 6
    PULSOUT servo,pulse
    PAUSE 10
  NEXT

'Get Sonar Data here & write it to EEPROM

  memAddress = 0
  FOR i = 1 to trackReq
    PULSOUT init, 5     '10 ms initpulse
    PAUSE 2
    RCTIME echo,1,R
    write memAddress, R.lowbyte
    write memAddress+1, R.highbyte
    memAddress = memAddress + 2
    PAUSE 20
  NEXT
  memAddress = 0
  GOSUB Alpha_Beta
  R = rs1 + (rOffset*scale)

  debug DEC R/scale,CR

  IF R<(maxRange*scale) then RangeOK
  R = maxRange*scale

  IF R>(maxErr*scale) then RangeOK
  R = maxErr*scale

RangeOK:
  WRITE memAddr2, R.lowbyte
  WRITE memAddr2+1, R.highbyte

  deg = deg - angleStep
  memAddr2 = memAddr2 + 2

  IF deg > minDeg THEN BGLoop

  FREQOUT piezo, 1000, dnote

'Wait for start button

  LCDptr = StartMsg
  GOSUB Write_LCD
Loop3:
  if StartBtn = 1 then Loop3:

'===================================================================================
=====
'
```

```
'Track - Track object, store coords in EEPROM. Goto alpha-beta when ready.
'
'===========================================================================
=====

Start_Track:
  trackNum = 0

New_Track:
  LCDptr = ScanMsg
  GOSUB Write_LCD

  GOSUB Init_Scan

Keep_Tracking:
  IF trackNum=0 THEN New_Track

  LCDptr = TrackMsg
  GOSUB Write_LCD

  number = trackNum
  GOSUB Num_LCD

  LCDptr = AngleMsg
  GOSUB Write_LCD

  number = deg
  GOSUB Num_LCD

  LCDptr = RangeMsg
  GOSUB Write_LCD

  number = R/scale
  GOSUB Num_LCD

  memAddress = (trackNum-1) * 2

  TargetX = TargetX + (maxRange*scale)    'X origin is -maxRange from base (makes x
always pos)

  write memAddress, targetX.lowbyte
  write memAddress + 1, targetX.highbyte

  memAddress = (trackNum-1+trackReq) * 2

  write memAddress, targetY.lowbyte 'Y origin is base
  write memAddress + 1, targetY.highbyte

  IF trackNum = trackReq THEN Det_Coords

  GOSUB Next_Scan
  GOTO Keep_Tracking

Det_Coords:

'Use Alpha-Beta to get x position & velocity
```

```
  memAddress = 0
  GOSUB Alpha_Beta

  targetX = rs1 - (maxRange*scale) + (baseX*scale) + (vs1*leadTime)  'Return to
rover home coord system

'Use Alpha-Beta to get y position & velocity

  memAddress = trackReq * 2
  GOSUB Alpha_Beta

  targetY = rs1 - (baseY*scale) + (vs1*leadTime) - yRetreat 'Convert to rover home
coord system

  targetY = targetY / scale

  IF targetX < 0 THEN Neg_X2
  targetX = targetX / scale
  GOTO Give_Coords
Neg_X2:
  targetX = (abs targetX) / scale
  targetX = -targetX

Give_Coords:
  IF targetY > roverXmax THEN No_Way
  IF targetX > roverYmax THEN No_Way      'Also excludes targetX<0, since its in 2-
compl

  roverX = targetX
  roverY = targetY

  LCDptr = RoverXMsg
  GOSUB Write_LCD

  number = roverX
  GOSUB Num_LCD

  LCDptr = RoverYMsg
  GOSUB Write_LCD

  number = roverY
  GOSUB Num_LCD

  LCDptr = RoverEndMsg
  GOSUB Write_LCD

'TRANSMIT TO ROVER HERE!

  FOR i = 1 to 10
    SEROUT xmitter, baud,["go", str roverX, str roverY ]
    PAUSE 20
  NEXT

  FOR i = 1 to 3
    FREQOUT piezo, 250, enote
    PAUSE 250
  NEXT
```

```
'Wait until START button pressed

Loop5:
  if StartBtn = 1 then Loop5:
  GOTO Start_Track

'If rover coords exceed rover limits, announce failure.

No_Way:
  LCDptr = FailMsg
  GOSUB Write_LCD

  FREQOUT piezo, 2000, cnote

'Wait for Start Button

Loop4:
  if StartBtn = 1 then Loop4:

  GOTO Start_Track

'=================================================================================
=====
'
'Main Scan: Find objects and report x,y coordinates
'
'=================================================================================
=====

Init_Scan:
  degStart = maxDeg

Next_Scan:
  GOSUB Setup_Plotter

  lastR = (maxRange - maxErr)*scale 'Set just under max range
  onTarget = no
  LOW onTargetLED

  deg = degStart
  GOSUB Calc_Pulse
  FOR i = 0 to 50
    PULSOUT servo,Pulse
    PAUSE 10
  NEXT

Scan_Loop:

  GOSUB Calc_Pulse
  FOR i = 0 to 7
    PULSOUT servo,pulse
    PAUSE 10
  NEXT

'Get Sonar Data here, limit range if reported if necessary
```

```
  PULSOUT init, 5 '10 ms initpulse
  PAUSE 2
  RCTIME echo,1,R

  R = R + (rOffset*scale)
  debug DEC R/scale,CR

  IF R < (maxRange*scale) then RangeOK2
  R = maxRange*scale

RangeOK2:
  memAddress = ((maxDeg - deg)/angleStep)*2 + (trackReq*4)
  READ memAddress, R0.lowbyte
  READ memAddress+1, R0.highbyte

'If significantly differnt from EEPROM, its a target.

 IF R > (R0 - (maxErr*scale)) THEN Off_Target

 IF onTarget = yes THEN Not_Done

  onTarget = yes
  HIGH onTargetLED
  R1 = R
  deg1 = deg

  FREQOUT piezo, 50, enote

Set_Sweep_Start:
  if (deg1+(sweep*angleStep)) > maxDeg then Sweep_From_Max
  degStart = deg1 + (sweep * angleStep)
  GOTO Not_Done
Sweep_From_Max
  degStart = maxDeg

 GOTO Not_Done

Off_Target:
 IF onTarget = no THEN Not_Done

  onTarget = no
  LOW onTargetLED
  FREQOUT piezo, 50, cnote

'Covert target range and azimuth to x,y coords

Calc_Target:

  deg = (deg1 + deg)/2
  deg = deg + (deg-90) + aoffset

  R   = (R1 + R)/2                  'Target is average of start and end range

  targetY = R */ (sin( deg*128/180 ) << 1)

  if deg > 90 THEN Neg_X
  targetX = R */ (cos( deg*128/180 ) << 1)            'Relative to base
```

```
  GOTO Count_Target
Neg_X:
  targetX = R */ (cos( (180-deg)*128/180 ) << 1)
  targetX =  -targetX                           'Relative to base

Count_Target:

'If object tracked enough times, go to alpha-beta filter to predict location

  trackNum = trackNum + 1
  RETURN

'If no target found, step to next location.

Not_Done:
  deg = deg - angleStep
  lastR = R

'If at the end of the sweep, and no target found, restart sweep from the start.

  IF deg > minDeg THEN Scan_Loop

  IF onTarget = yes THEN Off_Target 'If currently on target, assume end is here

  LCDptr = NoTargetMsg
  GOSUB Write_LCD

  trackNum = 0          'Reset track count

  RETURN

'==================================================================================
=====
'
'Alpha-Beta
'
'==================================================================================
=====

Alpha_Beta:
  READ memAddress, rs0.lowbyte
  READ memAddress+1, rs0.highbyte
  vs0 = 0

  FOR i = 1 TO (trackReq-1)
    memAddress = memAddress + 2

    read memAddress, rm1.LOWBYTE
    read memAddress+1, rm1.HIGHBYTE

    rp1 = rs0 + (vs0 */ dt)
    vp1 = vs0

    IF rm1 < rp1 THEN MakeNeg
      rs1 = rp1 + ((rm1 - rp1) */ alpha)
      vs1 = vs0 + ((rm1 - rp1) */ betaOverDT)
      GOTO Done
```

```
MakeNeg:
      rs1 = rp1 - ((rp1 - rm1) */ alpha)
      vs1 = vs0 - ((rp1 - rm1) */ betaOverDT)

Done:
    rs0 = rs1
    vs0 = vs1

  NEXT

RETURN

'===================================================================================
=====
'
'Setup Stamp Plot Lite as a radar scope.
'
'===================================================================================
=====

Setup_Plotter:

'RETURN
  DEBUG "!RSET",CR
  DEBUG "!TITL Sonar-Scope",CR
  DEBUG "!PNTS ",DEC 90/angleStep + 1,CR
  DEBUG "!TMAX 10",CR
  DEBUG "!SPAN 0,",DEC maxRange+baseY+maxErr,CR
  DEBUG "!AMUL 1",CR
  DEBUG "!CLMM",CR
  DEBUG "!CLRM",CR
  DEBUG "!TSMP OFF",CR
  DEBUG "!SHFT ON",CR
  DEBUG "!DELM",CR
  DEBUG "!SAVM OFF",CR
  DEBUG "!PLOT ON",CR

RETURN

'===================================================================================
=====
'
'Determine the length of pulse needed for servo from angle
'
'===================================================================================
=====

Calc_Pulse:

  pulse = (1000-500) * (deg-45)/90 + 500

  RETURN

'===================================================================================
=====
'
'LCD Subroutines
```

```
'
'===========================================================================
=====

'---------------------------------------------------------------------------
-----
'Initialize LCD for use
'---------------------------------------------------------------------------
-----

LCD_Init:
  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 LCD_Command
  char = %00000110                        ' inc crsr, no disp shift
  GOSUB LCD_Command
RETURN

'---------------------------------------------------------------------------
-----
'Convert number to ASCII and output to LCD
'---------------------------------------------------------------------------
-----

Num_LCD:
  zeroBlank = yes             'Starting, leading zeros can be
  FOR index = 2 TO 0          'Only outputing up to 3 digits
    char = number DIG index
    char = char + $30
    IF char = $30 then Handle_Zero
    zeroBlank = no
    GOTO Show_Digit
Handle_Zero
    If index = 0 then Show_Digit
    If zeroBlank = no then Show_Digit
    char = $20
Show_Digit:
    GOSUB LCD_Char
  NEXT
RETURN

'---------------------------------------------------------------------------
-----
'Write message to LCD
'---------------------------------------------------------------------------
-----

Write_LCD:
  READ LCDptr,char                        'LCDptr points to start of msg

  LCDptr = LCDptr + 1                     'Advance to next character
```

```
'If character is 0, it's done.
'If character is less then 32 or greater than 32, its a command to LCD
'Other wise, send it as text.
'Continue with next character.

  IF char = 0 THEN Msg_Done
  IF char < 32 THEN Its_a_command
  IF char > 127 THEN Its_a_command

  GOSUB LCD_Char
  GOTO Write_LCD

Its_a_command:
  GOSUB LCD_Command                        ' write the character
  GOTO Write_LCD                     ' go get it

Msg_Done:                            ' the message is complete

RETURN

'--------------------------------------------------------------------------------------
-----
'Send an individual command or character to the LCD display
'--------------------------------------------------------------------------------------
-----

LCD_Command:
  LOW RS                             ' enter command mode

LCD_Char:
  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

END


*********************************************************************************
```

11.2 Rover

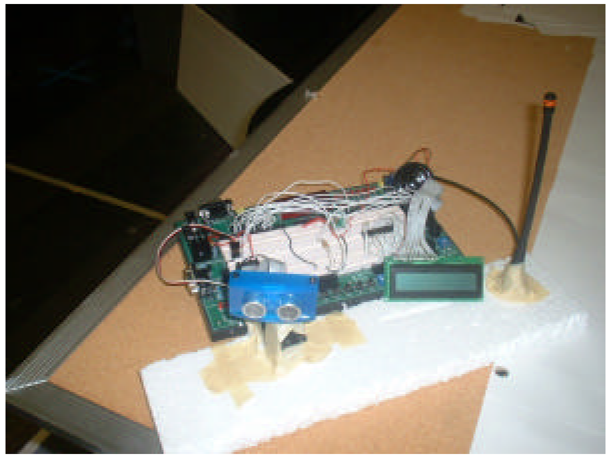
```
'{$STAMP BS2}
'======================================================================================
=====
'
' Final Project                   08 May 2003
'                                         MAE 576 Mechatronics
' LAB GROUP C
'                                                 FINAL VERSION!
' Rajani Boddu
' Craig Cole
```

```
' Sai Krishna Prasad Gavernini
' Preeti Sadanand Joshi
' Vikranth Bejjanki Rao
'
'=================================================================================
=====

'---------------------------------------------------------------------------------
-----

'Variables & Constants

pulse_count var word
x_dist      var         word
y_dist      var         word
temp        var         word
temp2       var         word
sensors var             nib
sense   var byte
baud        CON         17197
recdata1 var            byte
recdata2 var            byte
nt          var         nib

'=================================================================================
=====
'
'Initializaton
'
'=================================================================================
=====

low 12
low 13

main:
            nt =0
receiving:
            DEBUG "this is good", CR
            SERIN 3, baud, [WAIT("go"), recdata1, recdata2] ' Data received from
the base station

            DEBUG "x = ", DEC recdata1, CR
            DEBUG "y = ", DEC recdata2, CR

'This part of the code converts the co-ordinates recived into time for which the
servos need to
'be pulsed.
            x_dist = recdata1       'pulse time for x co-ordinate distance
            recdata1 = recdata1/5   'calibration
            x_dist = x_dist-recdata1
            'x_dist = x_dist - 20
            x_dist = x_dist*100
            debug "Left =", dec x_dist
            x_dist = x_dist*/$0014
            debug "Left =", dec x_dist
            x_dist = x_dist*/$2B7B
```

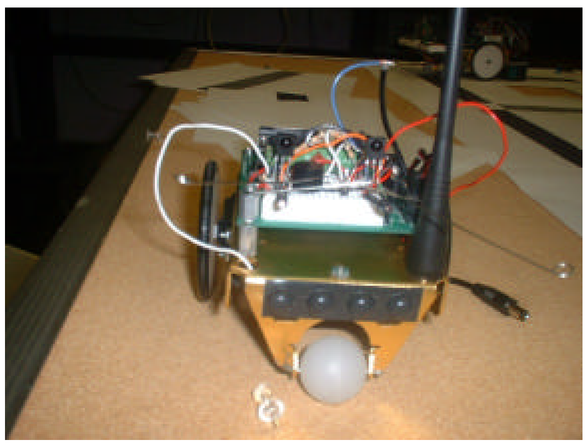
```
            debug "Left =", dec x_dist
            x_dist = x_dist/100
            debug "Left =", dec x_dist

            for pulse_count = 0 to x_dist 'the servo is sent pulses for the
computed time
                        if in6 = 0 then acquired 'checks if whisker has been
touched
                        pulsout 12,500
                        pulsout 13,1000
                        pause 20
            next

            gosub left_turns        'turns the rover left to start y-axis
traversing

            y_dist = recdata2       'pulse time for y co-ordinate distance
            recdata2 = recdata2/4   'calibration
            y_dist = y_dist-recdata2
            'y_dist = y_dist -35
            y_dist = y_dist*100
            debug "right =", dec y_dist
            y_dist = y_dist*/$0014
            debug "right =", dec y_dist
            y_dist = y_dist*/$2B7B
            debug "right =", dec y_dist
            y_dist = y_dist/100
            pulse_count =0

            for pulse_count = 0 to y_dist 'the servo is sent pulses for the
computed time
                        if in6 = 0 then acquired 'checks if whisker has been
touched
                        pulsout 12,500
                        pulsout 13,1000
                        pause 20
            next
            gosub sweeprimer
            gosub homing

homing:
            'Real Time homing

            if in6 = 0 then acquired 'checks if whisker has been touched
            freqout 7,1,38500
            sensors.bit0 = in8

            freqout 1,1, 38500
            sensors.bit1 = in0

            pause 18

            branch sensors,[forward, right_turn, left_turn, homesweep]

            goto obstacle_avoidance
```

```
forward:pulsout 13,1000: pulsout 12,500: goto obstacle_avoidance
left_turn:pulsout 13,500: pulsout 12,500: goto obstacle_avoidance
right_turn:pulsout 13,1000: pulsout 12,1000: goto obstacle_avoidance
homesweep:branch sense,[leftsweep,rightsweep]: goto obstacle_avoidance

'This part of the code implements the short range sweeping done by the rover

left_turns:
for pulse_count = 1 to 26
                        if in6 = 0 then acquired
                        pulsout 12,500
                        pulsout 13,500
                        pause 20
            next
            return

leftsweep:
            for pulse_count = 1 to 41
                        if in6 = 0 then acquired
                        pulsout 12,500
                        pulsout 13,500
                        pause 12
            next
            sense = 1
            goto obstacle_avoidance

rightsweep:
            for pulse_count = 1 to 35
                        if in6 = 0 then acquired
                        pulsout 12, 1000
                        pulsout 13, 1000
                        pause 12
            next
            sense = 0
            goto obstacle_avoidance


sweeprimer:
            for pulse_count = 1 to 24
                        if in6 = 0 then acquired
                        pulsout 12,1000
                        pulsout 13,1000
                        pause 12
            next
            return

acquired:
            output 2
            freqout 2,4000,3000
            goto main

stopping:'stopping and sleeping

stop
```

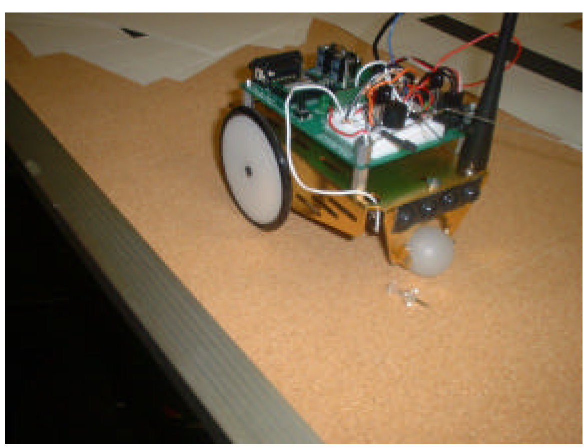
12. Picture Gallery



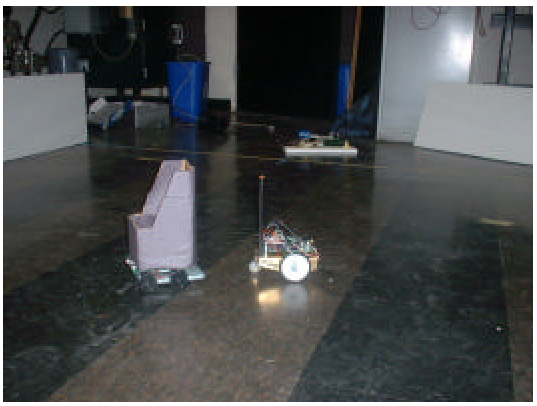
Base Station

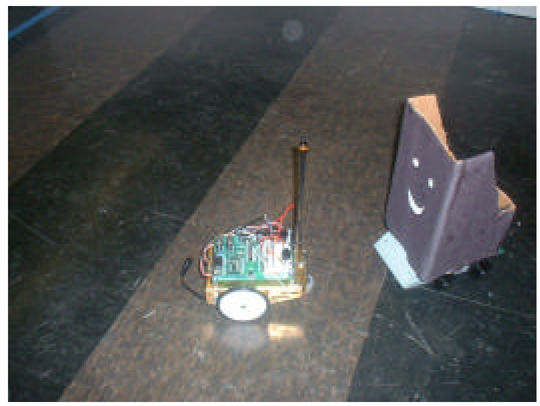


Rover



Arena





Intercepting the target