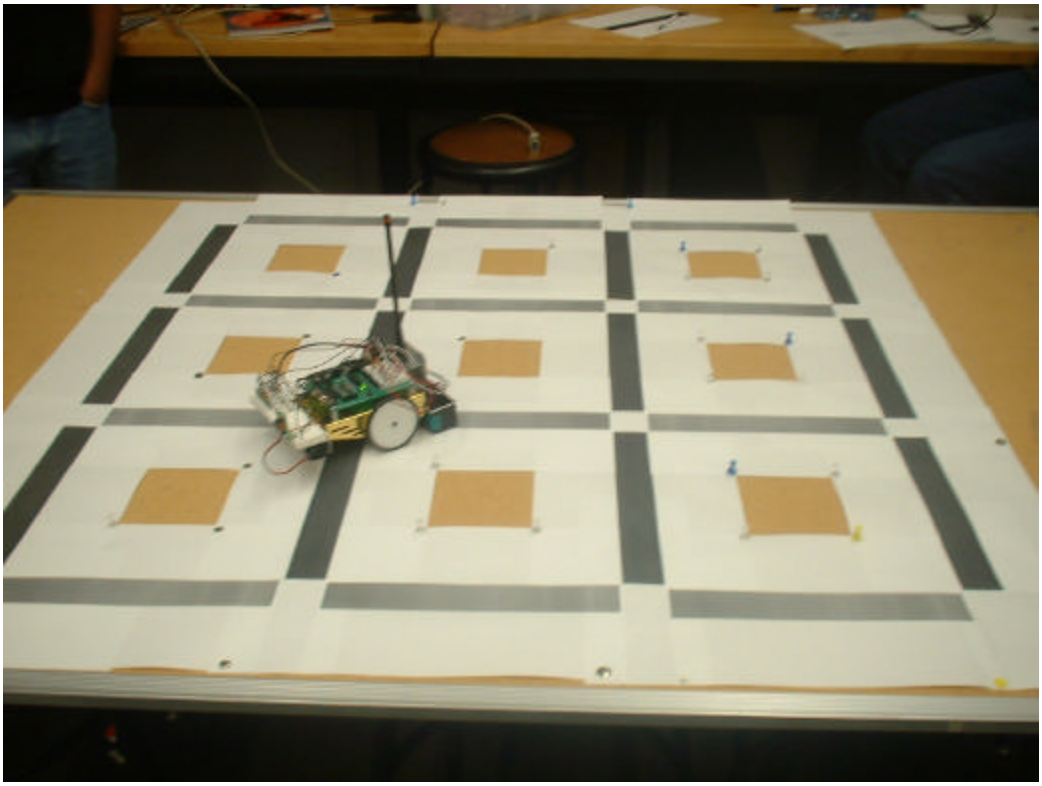
MAE 4/576 Final Project

Automated Guided Vehicle



Team E

Chetan Jadhav
Rohit Thali
Madan Mohan Reddy
Vijai Kalathur

Abstract:

This project was mainly about distributed control, computing and communication. This was to be accomplished by using a mobile robot and a base station. For us the mobile robot took on the form of an Automatic Guided Vehicle. Various modes of communication have been explored along with various hardware implementation possibilities.

Introduction:

In industries and many other useful applications, the use of automated vehicles for transport is an advantageous concept. Advantageous because these machines provide functionality unparalleled by what can be extracted from human labor. Some functions of an automated guided vehicle include:

- Transport of material from storage to work stations.
- Transport of partially finished/ fully manufactured goods to the storage from work stations.
- Automated vehicles also from part of the manufacturing line.

The automated vehicle does not act totally autonomously. In most industries automated vehicles are made to perform routine canned tasks, whereby they carry out one particular task endlessly. Such an example would be a vehicle servicing a fixed set of work stations, in doing so the vehicle follows a fixed path endlessly. The paths themselves could be of many kinds. A few examples are rails on the ground, differentially colored strips, overhead rails. The other kind of AGV's involve more human intervention, whereby an operator is more deeply involved in functioning of the AGV. We are trying to model an intermediate AGV, which involves automated features and at the same time allows operator intervention. This will allow us to make use of various kinds of sensors to operate the robot and at the same time require communication to mimic operator intervention. In the following sections we will delve into the details about how the project was implemented.

Details in the Project:

Hardware and Implementation:

We used two kits for this project, the BOE BOT Kit and the previously provided Stamp Works Kit. Using two kits together provided us with a lot of flexibility in terms of combination of hardware. We had alternative solutions available for key parts in our project. As an example, for navigating our robot, we had the option of using either an IR detector or photoresistors. Descriptions of all the components used can be found in the kit manuals. Here we are explaining some of the important components and their implementation along with some of the problems we faced.

Communication:

We used two modes of communication, RF Communication which the base station uses to send information to the mobile robot and the Fire-Stick II, which provides asynchronous serial communication to send information back from the mobile robot to the base station.

RF Communication:

RF Communication was carried out with the use of the TWS-434 RF transmitter, RWS-434 RF receiver, and two 433MHz Whip Antennas – one for the transmitter and one for the receiver. The component diagrams and connection charts are show below.

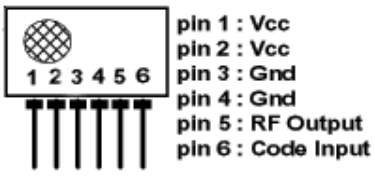


Figure 1: Transmitter for RF Communication.

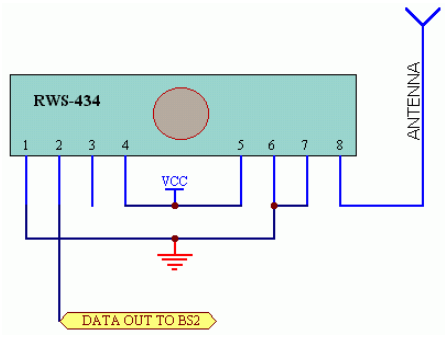


Figure 2: Receiver for RF Communication.

Pin Connection Chart for RF Communication:

Pin On TWS-434	Pin On BS2 – Base Station/ Connection
1 Vcc	Power Rail
2 Vcc	Power Rail
3 GND	Ground Rail
4 GND	Ground Rail
5 RF Out	Antenna
6 Data In	BS Pin 4
Pin On RWS-434	**Pin On BS2 – Mobile Robot/ Connection**
1	6, Gnd
2	Pin 7 BSII
4, 5	Power Rail
6, 7	Gnd
8	Antenna

Table1: Pin Connections for RF Communication

Implementing the RF Communication was an easy task. The only possible confusion was due to the synch, if this was similar to that used by other groups then there was a chance of operations being disrupted, but this did not occur and RF Communication was a peaceful task. The RF Communication was used by us to send signals to the mobile robot concerning the tasks it had to carry out. In our project the robot had to move from the base station to the desired intersection and stop there. To carry out this we first told the robot which station to move to using this mode of communication. The next command was to ask the robot to come back to the base station. This was again done by sending out a signal which indicated that the robot had to come back to the base station. The signals were basically numbers or characters, depending upon which signal was received the program would jump into the specific subroutine and carry out the required task.

IR Communication:

IR Communication was implemented using the Fire Stick II which is an asynchronous serial infra red communication device. We used IR Communication to send signals back to the base station from the mobile robot. The diagram bebw shows the IR signal receiver and transmitter.

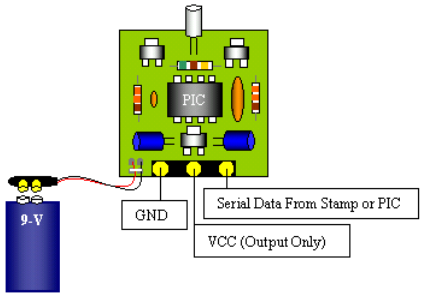


Figure 3: Fire Stick II Connection Diagram.

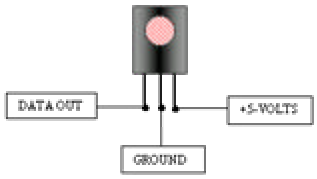


Figure 4: IR Receiver Connections.

Pin Connection Chart for IR Communication:

Firestick Pin	**BSII Mobile Robot Pin/ Connections**
Gnd	Ground Rail
Vcc	Power Rail
Data In	BS Pin 9
IR Receiver	**BSII Base Station Pin/ Connections**
Data Out	BS Pin 0
Vcc	Power Rail
Ground	Ground Rail

Table2: Pin Connections for IR Communication.

Implementing this mode of communication was again an easy task. One obvious problem we faced was the line of sight limitation. This meant in certain positions the signal sent out was not being received at the base station. We tried various alternatives and finally settled on mounting the receiver high above the ground, at a height of approximately 12 feet and pointing the transmitter towards the ceiling. This move solved all our problems and we were able to correspond from any position on our platform. When the robot reached the designated intersection it sends out a signal to the base station indicating the same. Again after traveling from the intersection back to the home station it sends out a signal. We also incorporated an obstacle detection feature, when an obstacle is detected the mobile robot comes to a halt, at this moment a signal is sent out to the base station indicating an obstacle in path. Similar to RF Communication, the actual signals being sent were characters or numbers which then caused the program to jump into relevant subroutines.

IR Ranging Sensor:

The base we used for our project represented a shop floor layout. There is always the possibility that the path used by the AGV might be blocked by an obstacle. To counter this problem we used the range detector. We used the Sharp GP2D02 ranging sensor for this purpose. This device is extremely small, as can be seen from the comparison below leading to ease of mounting.

Figure 5: Sharp GP2D02

We mounted the ranging sensor at the front side of our robot. The ranging sensor returns values between zero and two hundred and fifty. These values are inversely proportional to the distance between the sensor and the object from which the rays are being reflected back. In implementing it we kept checking the reading obtained from the sensor against a preset value. If the obtained value was below our checking value then the robot stood still on the spot and sounded an alarm so as to alert the elements blocking the path. Once the obstacle is removed from the path, the value obtained from the ranging sensor increases and rises above the check value. This causes the robot to resume its normal motion again.

RC Circuit:

After all the assembly of main components was done, we tested the robot. We found that the robot was exhibiting jerky and shaky movement, this was because of the RC circuit. Initially we were using a 0.01 micro Farad capacitor, the RC time was higher and the circuit was very sensitive. We decided to use a capacitor of lesser rating, to this end we tested the performance with a 0.001 micro Farad capacitor. We found that the robot was running smoother as compared to its previous motion. The RC time values obtained were less and the circuit consisting of the photoresistors was less sensitive.

Platform Setup:

The AGV in this project is setup to follow the black paths. All the paths (black lines) are 1.5 inches wide. The length of each path used is constant, but that makes no difference in our project as the AGV will keep following the black path until it hits an intersection, which is denoted using the white spot. The sensors will make sure that the AGV does not go out of path and uses the white spots or intersections to determine which direction it should go next. Although we only demonstrated our project for straight rectangular paths, the AGV will also follow paths that are curved or run in non linear paths. For the demonstration, we used each intersection as a drop off/pickup station which has a number. This makes it easier to control as opposed to giving specific coordinates every time. The home or warehouse station is station 0.

Fig 6: AGV in operation

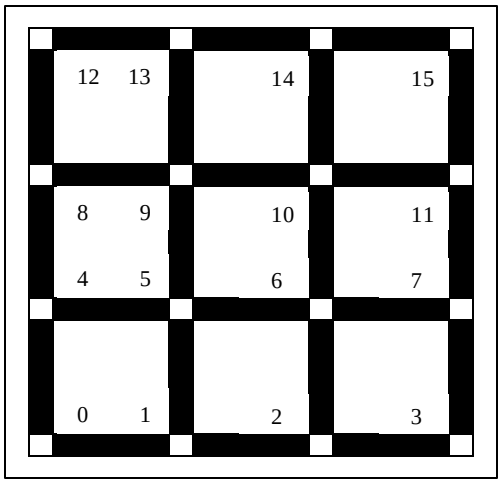
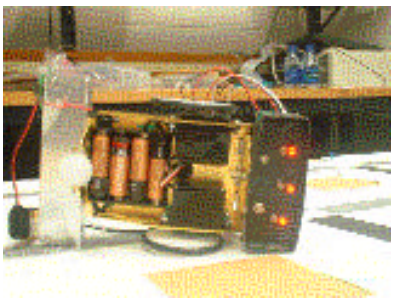


Fig 7: Grid

Fig 6, 7: An actual image of the path setup, Numbering of the stations

The Three Eyes:



Path Guiding Eyes

Consider a robot. We want to make it follow a line of certain width. For this, we need a minimum of two light sensitive sensors. This is because, both of them remain in the white part (the open road) when the robot is following the correct path. When the robot deviates, one of them comes into the black region (the guiding line). Then depending upon the sensor which came into the black region, we will send correction factor to the other that side (to the other motor) saying that move by so and so amount so as to get back in to the track. By this process of correction, what we will establish is that we have a self-correcting machine that will not come out of the black line region. To make the robot have a smoother motion (and to avoid heavy correction factors) we will keep the sensors just outside the thickness of the black line.

The light sensors what we are having here, are the LED's that emit light and photo resistors that will detect the light emitted by the LED in the amount based on what color the reflection is coming from. Initially we thought of using only the photo-resistors, but this idea had a defect. The photoresistors were too sensitive to the surrounding light. This necessitated us to close the sensors area, which meant we had to use some artificial light within the system for each photoresistor. This process of determining what was necessary for the sensors to work took 40% of our time for the project. This is because, testing of each new position was to be done to get a good grip and robustness for the system not to get altered even if the sensors were moved.

Middle Eye

Initially, we thought that the two sensors would work well if we keep an intersection. This is because, we thought that they both will enter the black region at that moment and we can take another conditional loop for counting the intersection. But, it turned out during testing the error that normally comes, when the robot moves, can make the robot end up in such a position, that it will take a left or a right turn depending at the intersection which depends which sensor moved into the black region. This was a very hard problem to realize immediately.

This forced us to use a third sensor and a new track design to adopt this third sensor. This sensor remains most of the time in the black region when the robot is traveling. When it encounters white (the cross-section spots are also made white for this third sensor) it will count the intersection and go to another loop which is an estimation of how much the robot has to be pushed forward till the middle sensor comes back into the black region, if we were needed to keep moving forward. If we needed a left turn or a right turn, necessary conditions have been implemented depending on the count which will make the robot take the necessary turn at that moment. The forward motion which I have just explained was necessary, because if we allow the robot to continue it takes many counts of the same white region. This should not happen and hence the prevention step. A 180-degree turn has also been implemented. All these work with utmost perfection in turning the robot, because of the third sensor. The only condition that has to be implemented was that the robot keeps moving until the middle one gets into the black region (+ some offshoot so as to give a little more robustness in correction of the path).

Finally, we have three LED's and three photo resistors all fixed close to the ground and enclosed so that outside light does not enter the system.

Execution/Flow of control of AGV:

The program first waits for the user to enter the station number that he wants the AGV to go to in the base station and this is communicated to the AGV wirelessly. If the user presses the go home button then we skip the rest of the main_main routine and go directly to the main routine. We have a look up table that associates each station number to a coordinate and we obtain the end x and y coordinates from this table. The AGV is initially at coordinates (0, 0). If the end coordinates are (0, 0) then the AGV stops and waits for the next input. There is a special case where the end coordinates are on the y-axis itself (i.e. x=0). We check for this case and do a u-turn if that is the case as the AGV is facing the x-axis by default. This is done in the main_main routine.

Then we enter the main routine where we first read the RC time of the three photoresistors. Then we check for obstacle detection. We used the idea of a semaphore here for this. If we checked for an obstacle every time in main, the AGV became very slow. So we used a variable sem that we increment by 1 every time in main. We check to see if sem mod 6 is 0 and if it is then we go into the obstacle detection and handling routines. We tried it for a bunch of numbers and we found 6 to be a good compromise between speed and good detection.

If no obstacle is found, we check if the value read by the middle sensor is less than 20. By our calibration, if the middle sensor reading is less than 20, it means it has detected a white spot which means an intersection. If it is greater than 20, we check if the absolute value of the difference between the left sensors's reading and right sensor's reading is less than 3. If it is not it means it is following the path properly and will go straight and we go back to main and repeat the process.

If the difference between the left sensors's reading and right sensor's reading is greater than 3, then we do path correction. Based on which sensor's RC time is higher we turn one pulse left or right and go back to main and repeat the steps.

If the value read by the middle sensor is greater than 20, then the AGV has come to an intersection and we need to evaluate what the AGV should do next. x_count keeps track of the number of intersections that the AGV has passed in the x axis and y_count keeps track of the number of intersections that the AGV has passed in the y axis. If the number of intersections that the AGV has crossed on the x-axis is equal to the end_x value it means the AGV only has to go on the y-axis after this. If this is the case then we increment the y_count. If both x_count and y-count are equal to end_x and end_y, then the AGV has reached its destination. The code for all the other conditions checked for here is given below. Explanation of the subroutines it goes to is explained later.

```
if x_count=end_x then y_inc
x_count=x_count+1
if x_count = end_x and end_y = 0 then check_pos2
if y_count = end_y and x_count = end_x then check_pos
if x_count = end_x and y_count <> end_y then left_turn
goto straight
y_inc:
y_count=y_count+1
if y_count = end_y and end_x = 0 then check_pos1
```

```
if y_count = end_y and x_count = end_x then check_pos
goto straight
```

The check_pos routine basically just turns the AGV a little to the left so that it is diagonally standing after reaching its destination. This makes it easier for to go back to the warehouse. It also sends out a message back to the base station saying it has reached it's destination and resets x_count and y_count so that it can go back home. The code is shown below for the check_pos routine.

```
for pulse_count = 1 to 55
pulsout 14, 600
pulsout 15, 750
pause 15
next
serout 9,17197,["P",x_count]
serout 9,17197,["L",y_count]

x_count=0
y_count=0
```

The check_pos1 routine is similar to the check_pos routine except it comes here if the end destination is on the y-axis itself (i.e. end_x = 0). If this is the case then the AGV takes a U turn and sends out a message to the base station that the destination has been reached. We then clear the end_y, x_count and y_count variables and end_x to end_y so that the AGV can get back to the warehouse easily.

```
gosub u_turn1
serout 9,17197,["P",x_count]
serout 9,17197,["L",y_count]

x_count=0
y_count=0
end_x = end_y
end_y = 0
```

The obstacle routine calls the read02 routine which reads the measurement from the range sensor. If the value read is greater than 110 than it means an obstacle is in the AGV's path and it keeps beeping until the obstacle is removed.

obstacle:

```
GOSUB read02 ' call measurement routine
if val02 > 110  then alarm:
goto noalarm

alarm:
freqout 6,100,2000
goto obstacle

noalarm:
return
```

The remaining methods, left_turn, right_turn, u_turn, u_turn1, and u_turn2 are all just simple routines that make the AGV turn in different directions. The reason for the three U turn routines is to make the U turn in different directions based on the location.

Source Code:

For the Rover:

```
'{$STAMP BS2}

'Code for the Rover
'----- Declarations --------------
left_photo var word       ' For storing measured RC times of
right_photo var word      ' the left & right photoresistors
mid_photo var word        ‘ middle photoresistor value
last_mid_photo var word   ‘ Storing the previous value of mid_photo
x_count var nib           ‘ X-axis coordinate
y_count var nib           ‘ Y-axis coordinate
photo_count var word      ‘ variable for storing the observed value
photo_count=0
mid_av var word           ‘ Used for getting absolute value
mid_av=0
sem var byte
sem = 0
x_count = 0
y_count=0
pulse_count var byte      ‘ variable for sending pulses to the motors
SYNCH CON "A"             'Establish synchronization byte
BAUD CON 16780            'N2400 baud (MAX)
DAT VAR byte              'Data storage variable
end_x var nib
end_y var nib
val02 var byte            ' value where reading is stored
i var byte                ' count variable for loop


'-----------------------------------------------------
' constant declarations

cl con 1                  ' pin 14 is output (clock)
dt con 2                  ' pin 15 is the input (data)


'-----------------------------------------------------
' I/O pin setup for detector

INPUT dt                  ' make pin 15 the input
HIGH cl                   ' make pin 14 output and high


'----- Main Routine --------------


main_main:                ‘ loop for intial communication (RF)

SERIN 7,BAUD,[WAIT("G"),  DAT]
if dat = 144 then main

‘ tables for getting the coordinates based on the station number
received

lookup dat,[0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3],end_y
lookup dat,[0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3],end_x
```

```
debug "Desired location coordinates ",cr," X ", dec end_x," Y ", dec
end_y
if end_x=0 and end_y=0 then main_main
if end_y > y_count and end_x = 0 then orient_y
goto main
orient_y:
gosub u_turn



main:
sem = sem + 1
' Measure RC time for right photoresistor
high 5 ' Set P3 to output-high.
pause 3 ' Pause for 3 ms.
rctime 5,1,right_photo ' Measure RC time on P3.

' Measure RC time for middle photoresistor


high 3 ' Set P3 to output-high.
pause 3 ' Pause for 3 ms.
rctime 3,1, mid_photo ' Measure RC time on P3.
'write memory_count,mid_photo

' Measure RC time for left photoresistor.
high 0 ' Set P5 to output-high.
pause 3 ' Pause for 3 ms.
rctime 0,1,left_photo ' Measure RC time on P5.


if sem //6 = 0 then cont_obs     ‘ obstacle detection if 6 times
goto no_obs                       ‘ obstacle is detected continuously
cont_obs:
gosub obstacel

no_obs:


if mid_photo < 20  then check_count
goto no_check_count
check_count:
if x_count=end_x then y_inc

x_count=x_count+1                ‘ increment x-axis coordinate
if x_count = end_x and end_y = 0 then check_pos2
if y_count = end_y and x_count = end_x then check_pos
if x_count = end_x and y_count <> end_y then left_turn

goto straight


y_inc:
y_count=y_count+1                ‘ increase y-axis coordinate
'pause 1000
if y_count = end_y and end_x = 0 then check_pos1

if y_count = end_y and x_count = end_x then check_pos
'debug "Ycount: ", dec y_count, cr

goto straight
```

```
no_check_count:


if abs(left_photo-right_photo) > 3 then check_dir




forward_pulse:                    ‘ loop for making the robot move
forward

pulsout 14, 550
pulsout 15, 950

pause 10



goto main
' Jump to either right_turn or left_turn depending on which RC time
is larger
check_dir:
if left_photo < right_photo then right_pulse
if left_photo > right_photo then left_pulse

'----- Navigation Routines -------

left_pulse:                      ' Apply one pulse to left then
pulsout 14, 650
pulsout 15, 650

goto main                        ' go back to main routine.
right_pulse:                     ' Apply one pulse to right then
pulsout 14, 850
pulsout 15, 850

goto main                        ' go back to main routine.

'----- right turn routine -------

right_turn:
mid_av=0
pause 1000
for pulse_count = 1 to 60        ‘ pulse out 60 times to make a right
turn
pulsout 14, 690
pulsout 15, 900
pause 30 'in ' go back to main routine.
next
goto continue

'----- left turn routine -------

left_turn:
serout 9,17197,["P",x_count]     ‘ sending xaxis coordinate to base
serout 9,17197,["L",y_count]     ‘ sending yaxis coordinate to base
```

```
‘ The code for making the robot turn left
for pulse_count = 1 to 55      ‘ pulse out 55 times to turn left
pulsout 14, 600
pulsout 15, 750
pause 15
next
goto continue

continue:

GOTO main



'----- straight Routines -------


straight:
serout 9,17197,["P",x_count]     ‘ sending xaxis coordinate to base
serout 9,17197,["L",y_count]     ‘ sending yaxis coordinate to base


‘ code for making the robot go straight for some time near the
intersection
mid_av=0
FOR pulse_count = 1 to 40
pulsout 14, 580
pulsout 15, 900
pause 40
next
GOTO main

'----- Adjustment Routines -------


check_pos:
for pulse_count = 1 to 55         ‘ adjusting the position keeping
the right
                                  ‘ wheel stationary
pulsout 14, 600
pulsout 15, 750
pause 15
next
serout 9,17197,["P",x_count]
serout 9,17197,["L",y_count]

x_count=0
y_count=0

goto main_main

check_pos1:
for pulse_count = 1 to 95         ‘ adjusting the position keeping
the left
                                  ‘ wheel stationary


pulsout 14, 600
pulsout 15, 750
next
gosub u_turn1
serout 9,17197,["P",x_count]
```

```
serout 9,17197,["L",y_count]

x_count=0
y_count=0
end_x = end_y
end_y = 0
goto main_main

'----- adjustment Routine 2 -------


check_pos2:
gosub u_turn2
serout 9,17197,["P",x_count]
serout 9,17197,["L",y_count]
x_count=0
y_count=0
goto main_main




obstacel:

GOSUB read02                    ' call measurement routine
if val02 > 110  then alarm:
goto noalarm

alarm:
freqout 6,100,2000         ‘ sound for the event
goto obstacel

noalarm:
return


read02:
LOW cl                  ' turn on detector for reading
rl:
IF IN2 = 0 THEN rl      ' wait for input high
SHIFTIN dt, cl, MSBPOST, [val02]
HIGH cl                 ' turn detector off
PAUSE 1                 ' let detector reset
RETURN

'----- U turn Routine 1 and 2-------


u_turn:
for pulse_count = 1 to 1000
pulsout 14, 600
pulsout 15, 600
high 3                   ' Set P3 to output-high.
pause 3                  ' Pause for 3 ms.
rctime 3,1, mid_photo    ' Measure RC time on P3.
if mid_photo < 20 then next_turn
next

next_turn:
for pulse_count = 1 to 1000
pulsout 14, 600
```

```
pulsout 15, 600
pause 15
high 3                    ' Set P3 to output-high.
pause 3                   ' Pause for 3 ms.
rctime 3,1, mid_photo     ' Measure RC time on P3.
if mid_photo > 20 then stop_turn
next
stop_turn:
return

u_turn1:

for pulse_count = 1 to 1000
pulsout 14, 600
pulsout 15, 600
pause 15
high 3                    ' Set P3 to output-high.
pause 3                   ' Pause for 3 ms.
rctime 3,1, mid_photo     ' Measure RC time on P3.
if mid_photo > 20 then stop_turn1
next
stop_turn1:
return

u_turn2:

for pulse_count = 1 to 1000
pulsout 14, 900
pulsout 15, 900
pause 15
high 3                   ' Set P3 to output-high.
pause 3                  ' Pause for 3 ms.
rctime 3,1, mid_photo    ' Measure RC time on P3.
if mid_photo > 20 then stop_turn2
next
stop_turn2:
return
```

Problems:

We faced a lot of problems in this project. Since our project was based on precise path control we faced a lot of problems with precision. A lot of the problems pertained to calibration as there were constantly changing environments. After trying for a long while with different sensors, we decided to use three photoresistors to determine the path of the AGV. The initial problem we faced was that every time we changed to a different location, even 10 feet away, the natural light varied enough for the calibration of the sensor to be way off. We fixed this by localizing the light which the photoresistor detected by using one LED for each photoresistor and covering the portion where the photoresistors were with black tape and paper. We still had minor problems, if the angle of the LED changed. But this was a lot easier to handle. It took us a while to find the parts for this and to calibrate it perfectly. Another common calibration issue we faced was that the AGV didn't function very well with low battery, but this was not as big a problem for us as it was for some of the other groups. We had even bigger problems when we tried to implement a system where the AGV could go from any point on the map to any other point. The program to do this is pretty complex and we tried for days to implement this. One obvious problem we faced was running out of memory. This was something we could have overcome. The biggest problem was the paths. We needed some way to identify where the AGV was at every point. We tried to use two different shaded tracks (One black and the other gray) for the horizontal and vertical axis. Since we were trying to detect three different colors (white, gray, and black) with one sensor (the middle one) we had narrow ranges for each color. But since the printouts were not perfect, if the sensor detected one spot that a just a little bit lighter or darker, it threw the whole calculation off and the whole program went nuts. Also the problems mentioned earlier about batteries and light made the problem even more complex. Another problem that made things even more frustrating was that we had to have a paper cover the photoresistors all the way to the bottom where it was touching the paths. If it was not all the way down to the ground, outside light would come in and throw off the sensor readings. After running the AGV on the same path a few times, the path became lighter as the photoresistor cover was scrapping it and we had to calibrate the sensor readings every now an then. Even though we tried very hard for days and nights, there were just too many things needed for AGV to work perfectly based on absolute mapping. The hardware was just not powerful enough to do this. As an aside, we came up with an algorithm that we tested on Matlab that determined the path from any point on the map to another other point, but we didn't know we could interface the Basic Stamp to work with Matlab. Knowing that would have made this project a lot easier to implement. Other than that the only other problem we faced was with the IR sensor not receiving values from all points in the map. For this we connected the IR receiver to a really long wire and attached it to a stick and put it near the roof of the room and it worked perfectly.

Conclusion:

Overall, this was the most interesting project in this course. We used all the skills we learnt in the first three projects to complete this project. Our initial goal was ambitious and we were unable to complete it due to limitations in the sytem we were working with. However, we still managed to complete everything we wanted to do in our revised goal successfully and our project worked perfectly as we had wanted it. We learnt a lot about controls, sensors, and calibration in this project and overall it was a very productive and informative project.

Contributions:

This project involved a lot of continuous and coordinated work. It did not need breaking up into sub parts and individual experimentation. Every part of this project was done as a group activity. We can not mark out contributions as every one was equally involved and participated to the best of his capabilities.

References/ Reading Mate rial:

- Basic Stamp Programming Manual.
- Robotics Student Work Book.

- RF Communication System:
  http://www.rentron.com/Files/ht-640.pdf
- Fire-Stick II
  http://www.rentron.com/Files/fs-ii.doc
- IR Ranging System:
  http://www.hvwtech.com/dnload/DIRRS_1_4.pdf
  http://www.acroname.com/robotics/info/examples/GP2D02-4/GP2D02-4.html