ROBOTIX
TECHNOLOGY ROBOTIX SOCIETY

ROBOCOP

(TUTORIAL)

INTRODUCTION

Problem Statement of the event:
The problem statement is to build a fully automated robot that acts as a law enforcer. This enforcer must navigate a set of rooms and apprehend and kill criminals (red) and avoid hostages (green). Points will be given on the basis of number or criminals killed and number of hostages avoided. As with all anti-terrorist operations this will be done under cover of darkness.

The Event requires the following:

- Basic Locomotion for which a differential drive will be suitable.
- Knowledge of Basic AVR coding and usage of microcontrollers.
- Knowledge of Basic Image processing, colour detection, stereo vision etc.
- Knowledge of USART and Serial Communication.
- Line following.
- Knowledge of Shooting and tackling Mechanisms.

Let us cover these points one by one.

MOTION OF THE BOT

Motion of the bot can basically be done by making a differential drive, which will be driven by a motor driver circuit, which gets Input from the microcontroller according to the processing done by the interaction of the bot with its surroundings.
For motion of the bot and differential drive we go to the following link:

Differential Drive

MICROCONTROLLERS

The Microcontroller is the device which will control the robot and it is what makes the robot “autonomous”.
Its understanding and working is of utmost importance.
For Understanding of microcontrollers and their working we refer to the following link:

Microcontrollers

IMAGE PROCESSING

The most important and unique part of this event is the Image Processing part involved in it. The use of Image processing is inevitable in this event and hence, knowledge of it is of primary importance.
In order to detect the criminals and the hostages we need the knowledge of image processing by which the difference in colour etc can be detected and hence it can be tracked whether it is a criminal or a hostage and accordingly shot, tackled or avoided.
The Basic Image processing tutorial is already provided on our website which helps to understand how IP (Image processing) is done right from the beginning, starting from very basic tasks like capturing an image and working on them for small purposes like creating a gray scale image, histogram, etc.
Video tutorials are also provided on our site which gives us a clear understanding

of how to write down the code and how exactly does the code run.
Refer to the Basic Image processing tutorial and video tutorials for basic image processing provided on our website for a good understanding of IP and then you can work with the techniques provided in this tutorial for having a good algorithm and a good base for approaching the problem statement with ease.
Click **HERE** for the module on Basic Image Processing . The video tutorials are given below:

Basic IP Tutorial_1

Basic IP Tutorial_2

The above you tube video contains video tutorials for basic image processing.

Let us start with some important techniques which can be used effectively in our event.
After going through the basic IP tutorial you are expected to understand the basic terms used in Image Processing.

For Basic OpenCV functions refer here:
http://www.cognotics.com/opencv/docs/1.0/ref/opencvref_highgui.htm
http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html
http://flowdesigner.sourceforge.net/wiki/index.php/OpenCV
http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_ImageProcessing.htm

COLOUR DETECTION

Colour detection can be done by setting suitable threshold on the BGR channels by first calculating your threshold value and then converting the image into a binary image and hence removing noises, if any, from you image.
For example, we can do the following:
Firstly, we take a still image from the camera and then we click on the part which we are interested in, we calculate the average value or the modal BGR value of a set of pixels near our point of click and decide that as our threshold. If, for example, the average or modal value (whichever we wish) of the blue channel of the pixels is Avg_blue then we can set limit such that pixels with blue channel

value Avg_blue (+ or -) 10 are converted to white while the others black, similarly we can set threshold for the green and red channels as well. We will now show how to click and get the value of the BGR channels in our image. For this we use the following function.

```
void on_mouse(int _event, int x, int y, int flags,void *)
{
if( _event==1 )
{
ax=x; // Value of the x-coordinate of point of click is stored in the variable ax
ay=y; //Value of the y-coordinate of point of click is stored in the variable ay
}
}

void cvSetMouseCallback( const char* window_name, CvMouseCallback on_mouse
void* param=NULL );
cvWaitKey(0);
```

The function cvSetMouseCallback is used for the purpose where the job is to be done and its arguments contain:
First argument is the name of the window in which we are clicking, second argument is the name of the function written above it (here the function is on_mouse) by which the x and y coordinate of the point of click is calculated and can be stored in some variable (here the variables are ax and ay).
cvWaitKey(0) is written after the function cvSetMouseCallback so that the image window remains there for us to click on it.

Instead of making a binary image, we can also mark the object with the help of a rectangle or a circle by drawing a rectangle or a circle in the image as done in the following videos:

OpenCV – Colour Tracking

Colour Based Object Tracking

By colour detection the object is identified and the coordinates of the pixels are noted, after this a circle or a rectangle can be drawn by the use of the following commands.

```
/*draw a blue rectangle*/
   cvRectangle(img,                    /* the destination image */
               cvPoint(20, 15),        /* top left point */
               cvPoint(100, 70),       /* bottom right point */
               cvScalar(255, 0, 0, 0), /* the colour ; blue */
               1, 8, 0);               /* thickness, line type, shift */

/* draw a red circle */
   cvCircle(img,                       /* the destination image */
            cvPoint(110, 60), 35,      /* centre point and radius */
            cvScalar(0, 0, 255, 0),    /* the colour; red */
            1, 8, 0);                  /* thickness, line type, shift */
```

The above algorithm is followed and a code is made the link for which is given below. Go through this code for a better understanding of the algorithm given above.

http://pastie.org/1124550

The running of this code is shown in the video, the link of which is given below:

http://www.youtube.com/watch?v=CkMf1EBhr4Q

The running of this code along with the demo run of detection of the criminals and the hostages is given below:

http://www.youtube.com/watch?v=2J4yl6k2sq0

Note: The exact shades of the criminals and hostages may be different.

For further reference on colour detection you can go to the following links:

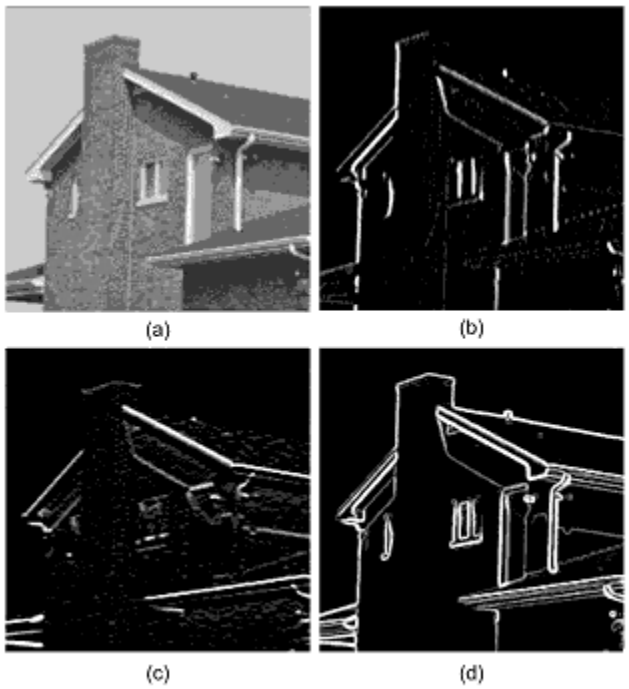
http://myopencv.wordpress.com/

EDGE DETECTION

Edge detection is a method to locate the edges of objects in an image. This can be very useful in this event as we can detect the edges of the criminals and hostages and hence detect their shape and size. This will be useful in their detection and appropriate actions can thereafter be carried out.
The Algorithm for this is quite simple and is as follows:

- Go through the image and analyze each pixel.
- For each of the pixel, analyze all the 8 pixels surrounding it.
- Among these look out for the value of the lightest and the darkest pixel.
- If the difference between the darkest and the lightest pixel is greater than a certain threshold value (which you are to decide depending upon what kind of image you are working on) then assign that particular pixel, value=255 (stands for white) else assign it to 0 (stands for black) in a new binary image created with same resolution as the original image.
- Display the binary image.
- It will be an image showing the edges of objects in the image.

Check out the edges on the following image for different threshold values:



We see that the images b and c are the images when proper thresholds are not set, while in image d the threshold is almost perfect. Hence we can hit ant try for setting up proper threshold and hence get a good threshold value.

SHAPE DETECTION AND PATTERN RECOGNITION

Shape detection is also a very important tool for detecting objects. Using shape detection we can get to know whether our object of interest is there in an image or not. For example, in this event the criminals and hostages are cylindrical in shape, so their projection on a 2 dimensional plane is a rectangle. Now by using shape detection, if in our image, we get a rectangular shape it may be a criminal or a hostage. By further detecting its colour we can conclude whether it is a criminal or a hostage or none of them and hence take necessary steps thereafter. The following algorithm is adopted for shape detection:

- The edge detection part is executed and the border line of each shape is recognized.
- The number of continuous edges in the image is counted.
- If there is a very sharp change in the direction of the line it means that there is a new line in the figure.
- The direction of the line is determined by determining the average vector between adjacent pixels.
- Now if there are four lines in this image then it is a rectangle.
- By measuring the angle between the pixels, we can determine more appropriately, what shape the image is.

The detection of a rectangular object in an image is very well shown in the image below, the edge detection code is first run and now the rectangle in the image is recognized and separated to work on it, the rectangle is shown here separately by a binary image. Similarly we can do in case of this event wherein we can detect the cylindrical objects as rectangles and separate them in our image and hence knowing their exact location on the screen.

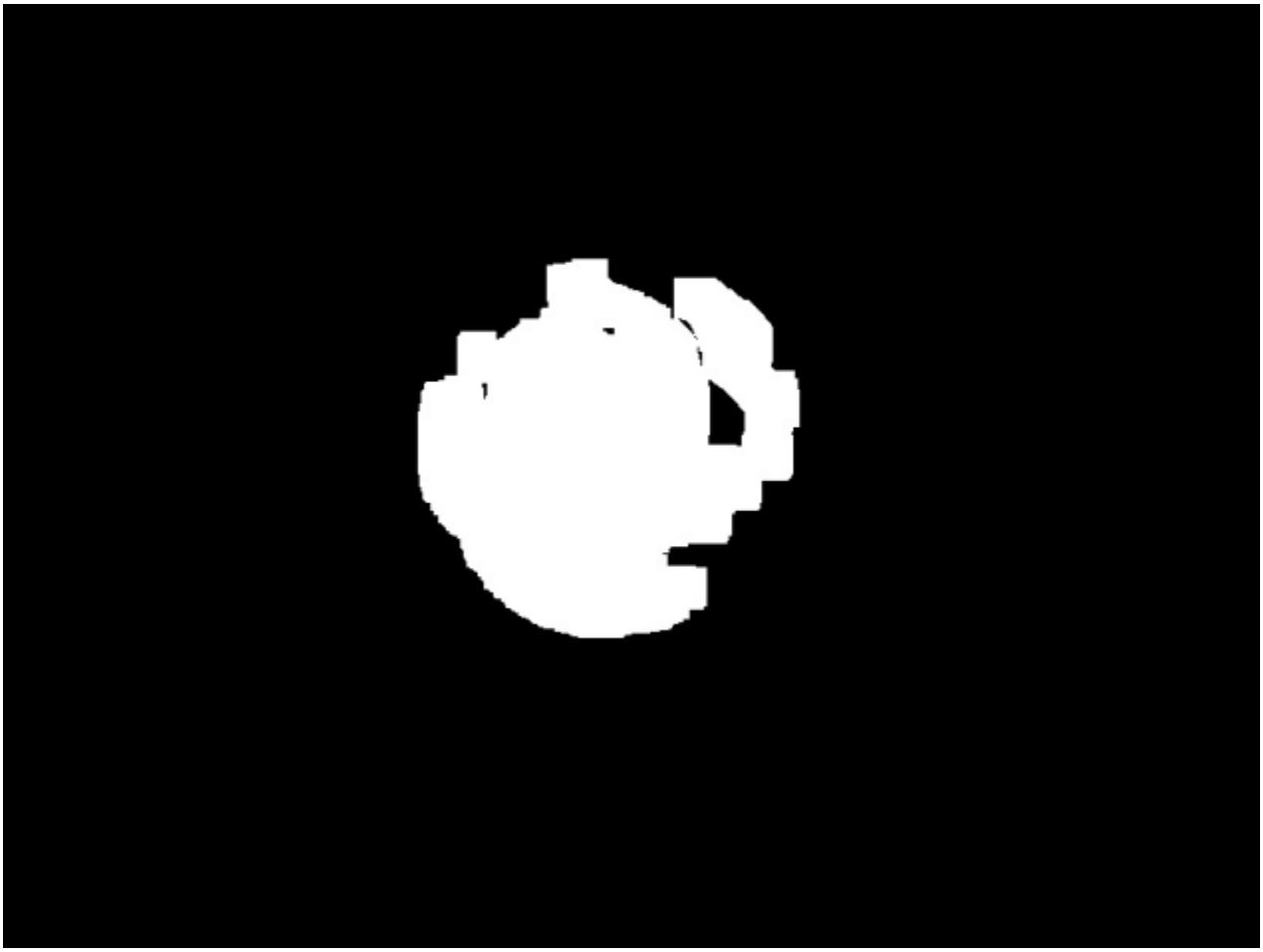
For multiple rectangles in an image, like for example if in a view we are getting from the camera, one criminal and one hostage is present, we get two rectangles in the image. What we can do now is mark these rectangles by numbering them one, two and so on. Now to detect whether it is a criminal or a hostage, we can look for colour inside the rectangle in the original image and hence come to know whether it is a criminal or a hostage and hence shoot them, tackle them or avoid whichever we want to do.
For reference we can look into:

http://www.emgu.com/wiki/index.php/Shape_%28Triangle,_Rectangle,_Circle,_Line%29_Detection_in_CSharp

MIDDLE MASS AND BLOB DETECTION

Another very important algorithm which can be very useful in our event is blob detection. Blob detection is basically used to determine if a group of pixels near each other are related to each other in some way or the other or not, for example it can be used to determine whether an object of a specific colour is present in the image or not.
Blob detection would be useful for detecting the criminals and the hostages because they will be a cylinder of a single colour hence we can identify them by this algorithm.

To find a blob we are required to threshold an image by a specific colour. Now that coloured parts in an image are converted to white and rest black in a binary image.
An example of blob detection is the following where the image is thresholded according to the red ball.



Middle mass is the centre of the blob calculated by taking the average of the coordinates of all the pixels in our image.
Finding middle mass can be useful in shooting the criminal so that if the aim is not very good, a small error will also result in hitting the criminal, as we are targeting the centre of the criminal.
Middle mass is sometimes not calculated very accurately when there are more than one blobs of the same color in an image.
To rectify this, we need to label each blob so that we can individually find the middle masses of each of them. The following algorithm is adopted in this case.

- Traverse the image pixel by pixel
- If the pixel encounters is of the color of the blob then we can mark it as 1.
- If the adjacent pixel is also of the blob color then label it as 1 again.
- If it is not the adjacent pixel then this means that a new blob is encountered.
- Label this as two now.
- In this way all the blobs will be labeled and hence the middle masses of each of them are found out individually.
- Repeat the steps until we cover all the pixels.

Here are a few videos on blob detection.

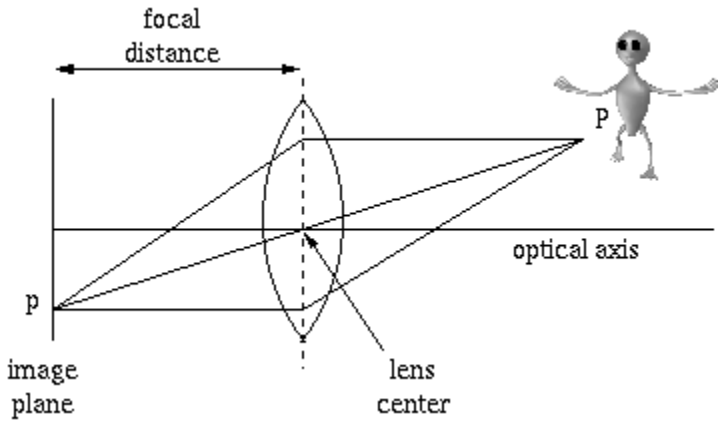
Blob Detection in OpenCV_1

Blob Detection in OpenCV_2

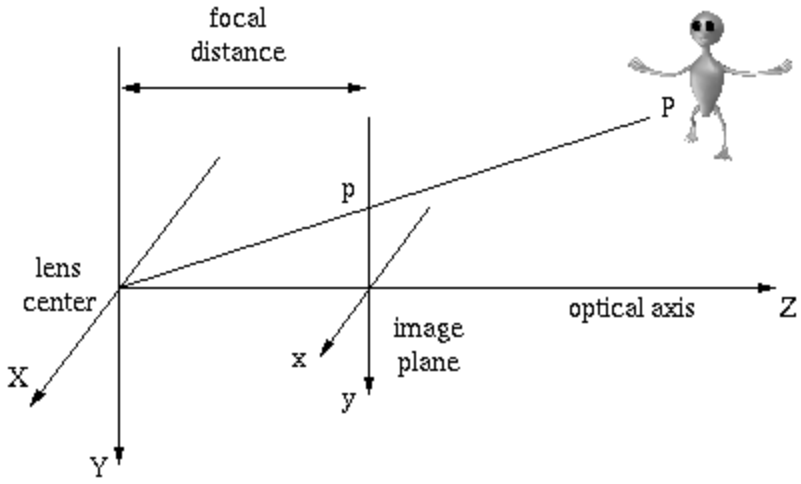
STEREO VISION

Stereo vision is a method of determining the 3D location of objects in a scene by comparing images of two separate cameras. Now suppose you have some robot on Mars and he sees an alien (at point P(X,Y)) with two video cameras. Where does the robot need to drive to run over this alien (for 20 kill points)?

First lets analyze the robot camera itself. Although a simplification resulting in minor error, the pinhole camera model will be used in the following examples:



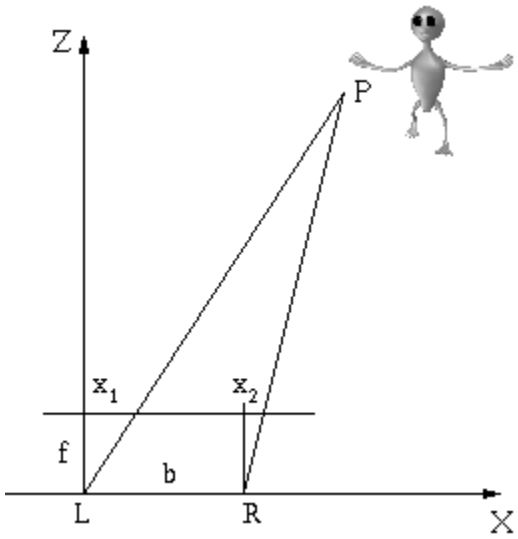
The image plane is where the photo-receptors are located in the camera, and the lens is the lens of the camera. The focal distance is the distance between the lens and the photo-receptors (can be found in the camera datasheet). Point P is the location of the alien, and point p is where the alien appears on the photo-receptors. The optical axis is the direction the camera is pointing. Redrawing the diagram to make it mathematically simpler to understand, we get this new diagram:



with the following equations for a single camera:

*x_camL = focal_length * X_actual / Z_actual*
*y_camL = focal_length * Y_actual / Z_actual*

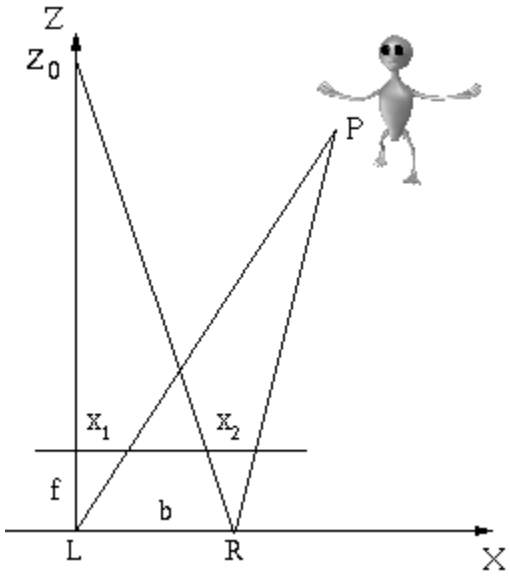
CASE 1: Parallel Cameras
Now moving on to two parallel facing cameras (L for left camera and R for right camera), we have this diagram:



The Z-axis is the optical axis (the direction the cameras are pointing). b is the distance between cameras, while f is still the focal length. The equations of stereo triangulation (because it looks like a triangle) are:

*Z_actual = (b * focal_length) / (x_camL - x_camR)*
*X_actual = x_camL * Z_actual / focal_length*
*Y_actual = y_camL * Z_actual / focal_length*

CASE 2a: Non-Parallel Cameras, Rotation About Y-axis
And lastly, what if the cameras are pointing in different non-parallel directions? In this below diagram, the Z-axis is the optical axis for the left camera, while the Zo-axis is the optical axis of the right camera. Both cameras lie on the XZ plane, but the right camera is rotated by some angle phi. The point where both optical axes (plural for axis, pronounced ACKS - I) intersect at the point (0,0,Zo) is called the fixation point. Note that the fixation point could also be behind the cameras when Zo < 0.



calculating for the alien location . . .

```
Zo = b / tan(phi)
Z_actual = (b * focal_length) / (x_camL - x_camR + focal_length * b / Zo)
X_actual = x_camL * Z_actual / focal_length
Y_actual = y_camL * Z_actual / focal_length
```

CASE 2b: Non-Parallel Cameras, Rotation About X-axis
calculating for the alien location . . .

```
Z_actual = (b * focal_length) / (x1 - x2)
X_actual = x_camL * Z_actual / focal_length
Y_actual = y_camL * Z_actual / focal_length + tan(phi) * Z
```

CASE 2c: Non-Parallel Cameras, Rotation About Z-axis
For simplicity, rotation around the optical axis is usually dealt with by rotating the image before applying matching and triangulation. Given the translation vector T and rotation matrix R describing the transormation from left camera to right camera coordinates, the equation to solve for stereo triangulation is:

p' = RT (p - T)
where p and p' are the coordinates of P in the left and right camera coordinates respectively, and RT is the transpose (or the inverse) matrix of R.
[Courtesy : http://www.societyofrobots.com]
In stereo vision the accuracy is a problem sometimes. If You are up for a challenge, and you are confident that you can do this with decent accuracy, you can try to implement this.
You can refer to the following links for understanding stereo vision and implementing it.

http://www.starlino.com/opencv_qt_stereovision.html
http://pages.cs.wisc.edu/~chaol/cs766/

http://www.tskills.com/default.asp?iID=MIFGL&item=MJIKD

http://cell.fixstars.com/opencv/index.php/StereoMatching

http://www.emgu.com/forum/viewtopic.php?f=2&t=247

http://sites.google.com/site/jeffcolombe/home/stereo-vision-with-webcams-in-opencv

LINE FOLLOWING

Since a line is given from the entrance door to the exit door in each of the rooms, these lines can be very useful in going for one room to the other. This task (going from one room to the other) would have otherwise been very difficult.
We can also avoid line following and just use the detection of the door colour (because the door is different in colour from the walls) for going from one room to the other, but it would be a very difficult task in that case, still if we can do that, we can save a lot of time in the process of going from one room to the other and hence score high.
Line following can be done by both, image processing as well as by the use of sensors.

LINE FOLLOWING BY IMAGE PROCESSING:

The accuracy of the line follower largely depends upon the efficiency of the algorithm used. We have two different algorithms which can accomplish the given task by different approach. The first one takes into account the curvature of the path as seen by the camera. The other one identifies the path as a white patch in midst of black background. First step is to remove as much noise as possible. Then the part of patch seen on the screen is divided into four levels. Highest priority is given to the bottom level. At each level the centre of the path is calculated. Each level had its predefined threshold which means if the path lies in this zone the bot is instructed to move straight.
Step by Step analysis of the algorithm:

- Firstly a suitable threshold is decided so that the line appears white and all the other objects appear black in a binary image.
- We try and remove the noise completely and get a clear binary image with the line as a white patch and the rest of the objects black.
- We now take 20 portions of the line grouped into four sub groups each having five portions.
- Each portion is a row in the binary image having considerable length of continuous white pixels.
- Now to each group we find the centre of area.
- If centre of area of the lowermost group is too much deviated from centre, the bot is turned in corresponding direction so that the road comes in centre.
- If it doesn't get much deviated, then the deviation of the second lowermost centre of area with respect to the lowermost is considered.
- If deviation is above a certain value of threshold we considerably turn the bot in particular direction.
- Now if both the centre of areas is within their respective threshold we move the bot forward.
- If we find no road in the second lowermost area or no road in the upper areas, we turn bot in the direction where we find the road in the lowermost area.
- If we find no road in even the downward one, the bot moves in reverse direction to get the road back on scene.
- This makes the bot move in the correct direction and follow the line.

The sample code is given here:
http://pastie.org/1122109

LINE FOLLOWING BY USE OF SENSORS:

Line following can also be done by the use of sensors. Using an LED-LDR sensor is very common for line following.
For line following using sensors we can refer to the following link:

Line Following

USART and Serial Communication

Communicating your computer and microcontroller consists of two parts, firstly, the part where the code is to be written and burnt in the microcontroller and the secondly, the part where the code is to be written and included in your project in your computer.

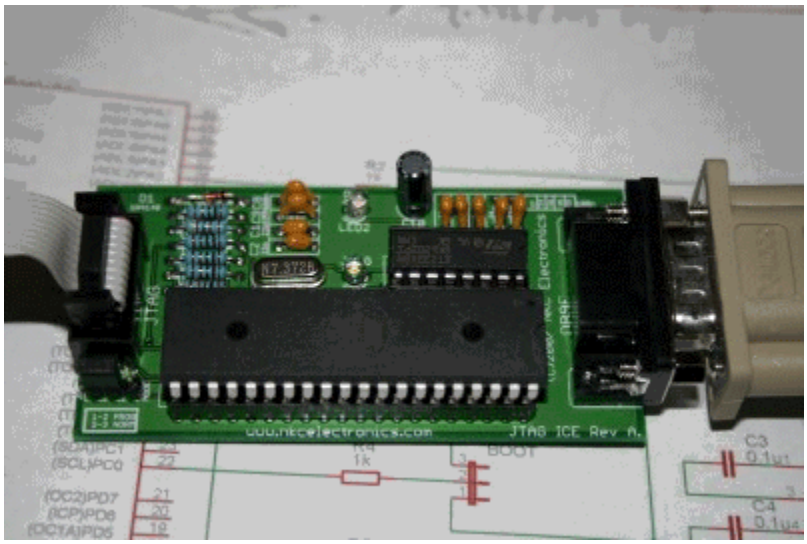
USART

USART stands for Universal Synchronous Asynchronous Receiver Transmitter, it is a standard and simple protocol for serial data transfer.

MAKING CONNECTIONS:

The following connection is to be made for serial communication, which is connecting the development board with the computer by a USB to serial port which looks like the following.

The serial part of the port is connected to the development board and the USB part is connected to the computer.
The connection is shown below:



CODING:

Codes are required to be written and burnt in the microcontroller.
Create a new project and add the following files to your project:
1.usart.h
2.usart.c
These files are provided for download on our website, in the ZIP file named "serial communication"
The link for which is given below:
[Direct Download]

Adding these files to your project is very important, without this USART won't work. These files are important in the functioning of serial communication.
After adding those files to your project, write down your code, a sample code is given here which does the following job:
If '1' is received, it switches all LED on
If '0' is received it switches off all LED
If 'a' is received it sends back "Hi" string to the computer
If 'b' is received it sends back "Bye" to the computer.
The code is given below:

Some important commands used are:

USARTInit(UBRRVALUE);[for initializing usart. The number to be entered in parenthesis is the UBRRVALUE. Which is 103 for 9600 bits per second,68 for 14400, and 51 for 19200 Bits/second]
UDataAvailable(); [To get number of data bytes waiting in queue to be called]
UWriteData(char data); [To send data to PC]
UWeiteString(char *str); [To send a string to PC]
UReadData(char data); [To read Bytes from a queue call]

THE SAMPLE PROGRAM:
http://pastie.org/1186128

```
#include <avr/io.h>
#include "USART.h"
void main()
```

```
{
char data_entered;//Byte received from USART
//Initialise USART with 19200Kbps
USARTInit(51);
//Initialize LED port as output
DDRC|=0b00001111;
while(1)
{
//Wait till data is available
while(UDataAvailable()==0);
//Read the data
data_entered=UReadData();
//Processing according to the data entered.
switch(data_entered)
{
case '1':
//Switch on all LEDs
PORTC=0b00000000;
UWriteString(" <All LEDs are now switched on>");
break;
case '0':
//Switching off all LEDs
PORTC=0b00001111;
UWriteString(" <All LEDs are now switched off>");
break;
case 'a':
case 'A':
//Write back to serial port
UWriteString(" <Hi>");
break;
case 'b':
case 'B':
//Write back to serial port
UWriteString(" <Bye>");
}
}
}
```

After Writing the above code compile the project and burn it in your microcontroller. Precompiled hex files ready to burn will be in the “precompiled-hex-file” folder.Burn this to your microcontroller.

The part to be included in your project in the computer.

The following files are to be included in your project in your computer along with the main program body written in the project.
The files are :

1. tserial.h
2. tserial.cpp
3. bot_control.h [File containing the definition of functions used in the sample program like, sendData(), startDevice(), stopDevice() and so on .]
These files are to be copied into the project folder and then added in the project. These are very necessary for the functioning of serial communication because they contain all the settings needed for serial communication to work.
These files are available on our site for download along with usart.h and usart.c in the zip file “serial communication”.
The link for which is given below:

<Link for downloading the serial communication zip file>

Will be Updated soon.

Now a sample program is written below. This is the main code of your project. It is a sample program for sending one character through serial communication.

```
#include <stdio.h>
#include <iostream>
#include <conio.h>
#include <stdlib.h>
#include "tserial.h"
#include "bot_control.h"
serial comm;
int main() {
char data;
printf("enter character to be sent");
scanf("%c",&data);
comm.startDevice("COM2", 9600)
/* “COM 2” refers to the com port in which the USB to SERIAL port is attached. It is
shown by right clicking on my computer, then going to properties and then device manager.
9600 is the baudrate in bits per second. */
comm.sendData(data);
comm.stopDevice();
return 0;
}
```

To understand these functions sendData(), startDevice(), stopDevice() go through the file bot_control.h thoroughly.

The above procedure is to be followed completely in order to send any information (a byte in this case) from your computer to your microcontroller by serial communication.

SHOOTING AND TACKLING MECHANISMS

SHOOTING MECHANISM:

A good knowledge of shooting mechanism is very important in this event, because shooting a criminal down will be very effective in scoring points. The criminal can be tackled as well but shooting is more effective as it is time saving and it does not require the robot to go off the line, to the criminal and tackle it down. In the process of leaving the line, coming back on it may be sometimes difficult, so we prefer to shoot down the criminal while standing on the line itself. There are a lot of mechanisms which can be used for shooting. A few of these mechanisms are very well explained in the following link:

Shooting Mechanism

While making the shooting mechanism we should take care that the mechanism used does not cause any harm to the surroundings.

TACKLING MECHANISM:

For tackling we have to first locate the exact position of the criminal and then go to it, and push it off by some means and make it fall.
We can tackle the criminal in the following ways:

- By going in front of it and pushing it by some sort of a rod coming out of the robot.
- By running the bot over the criminal.

- By blowing air at high velocity after going in front of the criminal.

These are some of the examples of tackling mechanisms, you can use any other mechanism of your choice if you wish to.

The demo run of shooting a criminal down or tackling it down is given below:

http://www.youtube.com/watch?v=ewr5tCnMKis

CONCLUSION

If you still have any doubts in understanding any part of the tutorial or any doubts regarding the event, you can post it on our forum.
The link for the forum is:
http://www.robotix.in/forum

FORUM

In case of any doubts regarding the rules of this event, feel free to post on the forum of our website http://robotix.in/forum

REGISTRATION

For registration, visit http://www.ktj.in/register/reg.php

HEAD

Rahul Das

rahul@robotix.in

+91 - 9775552399