REAL-TIME MACHINE VISION BASED ROBOT

A Novel Research Project Report

G. Sathya
Senior Lead
Automatic Data Processing, New Jersey, USA
E-Mail: sathyag413@gmail.com

Abstract

This paper discusses the practical implementation of a real-time machine vision based autonomous mobile robot. The relevant image processing concepts are implemented programmatically and the control of the servo motors is done using the Basic Stamp 2 microcontroller. Some minor concepts regarding the serial-port or RS-232 communication, and block diagram are discussed.

Keywords: AMR, Machine Vision, Target Tracking

1. Introduction

1.1 Experimental requirements

The requirements for this project are divided into two categories. One is the hardware requirements and other is the software requirements. Both are equally important when it comes to robotics based on machine vision. The requirements are as listed below.

Hardware requirements and their specifications

- ➢ Parallax Boe-Bot Robot Kit
  - Basic Stamp 2 microcontroller
  - Board of Education
  - Robot's aluminium body
  - Two Parallax servo motors
  - Four 1.5V AA sized battery
  - A RS232 serial cable
- ➢ A desktop or laptop PC
  - *Processor*: Intel Pentium4 or higher / AMD Athlon 4400+ dual core or higher
  - *Hard-disk*: 20 Gb or higher – preferably SATA drive
  - *RAM*: 512 Mb or higher with good bandwidth for data transfer
  - *Motherboard*: A RS232 serial port and at least two USB ports should be available
- ➢ A wireless or wired USB web camera
  - *Resolution*: 320x240 (minimum) or higher
  - *Video frame rate*: 10 – 15 frames/sec or higher
  - *Focusing range*: 30mm to infinity
  - *Colour*: 24 bit (RGB) or higher
  - Length of wire that connects the camera and PC should be at least 2 meters in case of wired webcam
- ➢ A USB to RS-232 converter (In case if the PC doesn't have a serial port)
  - Length of wire should be at least 2 meters

Software requirements and their specifications

- ➢ Operating system
  - Windows XP 32 or 64 bit

- Service Pack 2 or 3

- Visual Studio .NET 2008 with MSDN
- DirectShow.NET API with AForge.NET wrapper
- Webcam device driver
- Driver for USB to RS-232 converter (In case if the PC doesn't have a serial port)

These are the minimum hardware and software requirements as far as this robot is concerned. Use of advanced image processing and pattern recognition techniques might require high end hardware to process each frame if the frame rate is a matter of concern.

1.2 The Block diagram

The block diagram illustrates the various stages and sub stages that are involved in the system. The block diagram also illustrates the closed loop nature of the system.

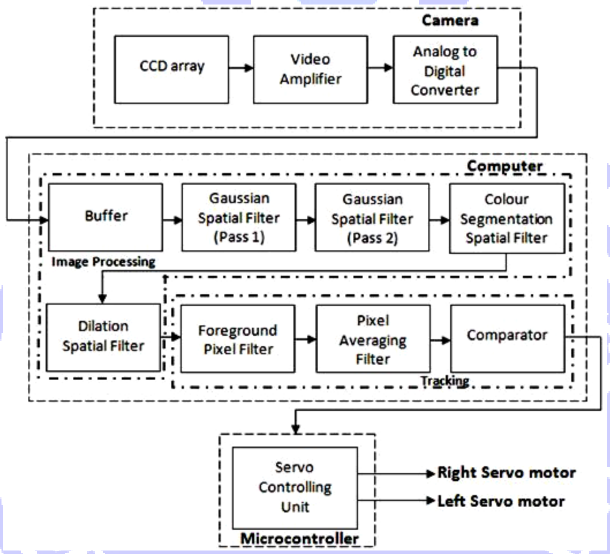


Figure 1.2.1: Block diagram of the machine vision robot system

The CCD (Charged Coupled Device) array is a semiconductor device, which captures the incident light and creates corresponding current as the output. Since the output of the CCD video signal is very low, it is to be amplified. Hence the amplifier strengthens the video signal that comes from the CCD. Since the video signal is purely analog in nature, it is converted to digital by the analog to digital converter. All these three stages reside within the camera. Hence a dashed box that encapsulates these three stages is shown in the figure 1.2.1.

After the conversion of the analog video to digital, it is time to process it. Since we have designed image processing filters that are meant to be used in spatial domain, we need to convert the incoming digital video signal to a signal that can be represented in spatial domain. Hence we introduce a digital buffer memory that stores the incoming signal in each memory location having their respective addresses. Each memory location represents a pixel's location and intensity value. Hence in this manner, the signal is now a spatial signal on which spatial filters can now be applied. The buffer also acts as an accumulator for the video frames if the digital spatial filters are no synchronous with the video camera. The spatial Gaussian filter is of 3x3 size and having an impulse response sequence as discussed in chapter 4. Since first pass doesn't prove to be effective on the noise which is Gaussian, the spatial signal is passed through another cascaded Gaussian filter that helps in reducing the amplitude of the noise further and thereby making it negligible.

The colour segmentation spatial filter can be compared to a bandpass filter for a 1-D signal. Each colour in an image can be attributed to a unique frequency of light. For example, colour red in the visible region of the electromagnetic spectrum is of lowest frequency and the colour violet has high frequency. The colours between violet and red have frequencies that lie between the frequencies of red and violet. The colour segmentation filter allows certain frequencies of light that are of our interest and rejects the rest.

The range of colour values in this case belongs to the object that is to be tracked. Since the light distribution over the body of the object is not constant, a small range of values are taken such that its variation is small. The word segmentation implicitly tells that the object is discriminated from the background on the basis of colour. Hence the output of this filter is a binary image or frame. The object in the image is represented by white pixels as foreground and the rest by black pixels as background. Hence, we'll be getting a stream of binary images or frames from the output of the colour segmentation filter.

Next comes the dilation filter that operates on the incoming binary images. As discussed in chapter 3, dilation filter grows the image. In this process of growth, the gaps or holes on the object which represent the background are eliminated so that the object looks like a continuous rigid body. This makes tracking easier. Image processing ends here and the tracking stage comes next.

In the tracking stage we make use of a filter that extracts the foreground pixel co-ordinates. Next, the foreground pixel co-ordinates which represent the object are averaged in the pixel averaging filter where the average of all the co-ordinates of the pixels that represent the body is calculated to give the centroid of the object. This is done for each frame.

The comparator compares of the centroid with a reference and gives the error of the relative position of the object with respect to the reference. Errors can be positive or negative along both x and y axes. But here, we consider only error along x-axis. If the error is negative, then the centroid of the object is on the left of the image or frame with respect to the reference co-ordinate and vice-versa when the error is positive. So the output of the comparator is only the error and its direction, which is a 1-D signal.

All the image processing and tracking operations are performed on a computer programmatically which is represented by a dashed box in the figure 1.2.1 encapsulating both image processing and tracking stages.

Once the servo controlling unit receives the error signal from the computer (from comparator in other words), it sends appropriate control pulses to the servo motors. Based on the direction, the desired servo motor will be activated. As outlined earlier, the control pulses are nothing but PWM pulses.

1.3 The Experimental Setup

The experimental setup deals with the arrangements can connections that are made in this project. The three main equipments in this project are the camera, the robot kit and the computer. The assembling of the robot kit after its purchase is not mentioned as it is already covered in its manual. Figure 1.3.1 shows the setup required for object tracking.

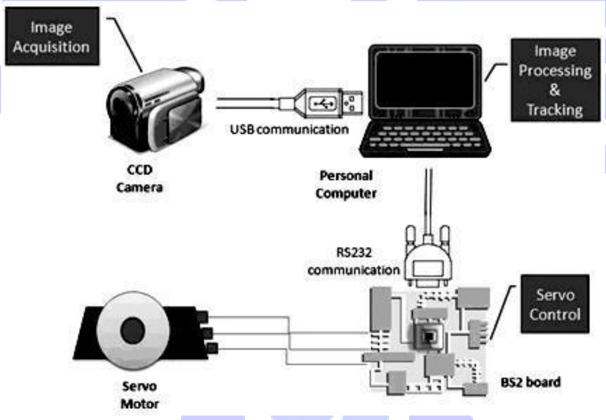


Figure 1.3.1: Setup for object tracking

The camera is physically attached to the front side of the robot. However, in the figure 7.3.1 the camera is shown separately for our understanding purposes. The camera is connected to the PC using a special serial data bus called the Universal Serial Bus (USB).

The BS2 microcontroller that resides on the robot is interfaced to a serial RS-232 port. Hence the connection between the microcontroller and the PC is done using the RS-232 serial communication cable. Again for the sake of understanding the connections, the microcontroller and the servo motors are all shown separately in the figure 1.3.1 though they are fixed to the robot.

2. Image Processing and Tracking Using PC

The video stream that comes from the camera is fed to the computer through USB port. The video stream is first decoded and decompressed digitally. The encoding depends on the type of the camera under use. Cheap cameras have low quality CCD compared to the expensive ones and thus the compression that is used also would vary accordingly. The bandwidth of USB communication is 1.5 to 480 Mbps. The bandwidth in turn depends on the type of USB controller used. Cheap USB controllers do not support high bandwidth.

The calculations for a video of resolution 640X480 are given in table 2.1.

Table 2.1: Calculation for 640X480 video resolution

Input Parameters	
Pixel size:	640 x 480
Frame rate:	30 FPS
Colour depth:	8 bit per colour
Calculation	
One frame Uncompressed size:	921.6 KB
Moving Image Pixel rate:	9.22 Mhz
Uncompressed bitrates:	221.18 Mbps or 27.65 MBps
Required storage:	1 second: 27.65 MB
	30 seconds 829.44 MB
	1 minute: 1.66 GB
	5 minutes: 8.29 GB
	1 Hour: 99.53 GB

From the above calculations, an uncompressed video needs large memory (frame buffer) for temporary storage and large data transfer bandwidth. If the resolution is high, then we may expect more memory and bandwidth. Hence it is required to compress the video stream rather than raw data. If the compression algorithm is lossy, then we may lose some high-quality details. Some cameras support compressions like MPEG4 and other do not. Usually, an uncompressed video is of lower resolution. Hence, one has to go in for a trade off when it comes to compression or resolution.

The memory that is used to temporarily store the frames is called *Frame Buffer*. Typically the frame buffer is stored in the memory chips on the video adapter. In some instances, however, the video chipset is integrated into the motherboard design, and the frame buffer is stored in general main memory. The main memory is nothing but RAM present inside the computer. Figure 2.1shows how RAM is used as frame buffer.

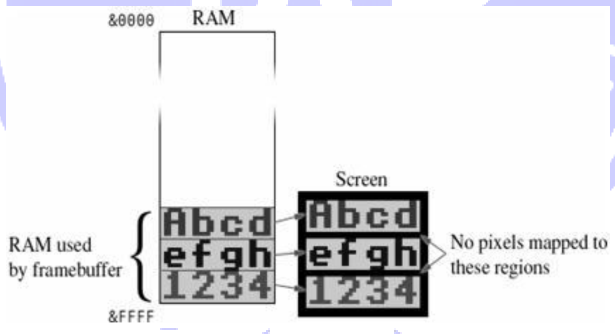


Figure 2.1: RAM used as frame buffer

Now that we know where the frames are stored, we need to access it programmatically. Before we access it, we need to make an event program that can generate events based on the fact that a frame is ready. If one frame from the camera has arrived, it will be stored in the frame buffer, and from there, the frame will be rasterized on to the screen.

After the new frame being appraised, it has to be converted to a bitmap image for application of different image processing filters. This whole process is termed as *Frame grabbing*. A software-based frame grabbing is used in this project instead of hardware. Frame grabbing can be done by using an API called *DirectShow*, which was developed by Microsoft and was included in earlier versions of a package called DirectX, which is a collection of APIs used for graphics and imaging. An API is abbreviated as Application Program Interface which is a set of functions, classes, protocols, interfaces etc, that are used by applications or programs. For example, the Windows API called *WIN32* is used by almost many user programs like Microsoft Word, Excel, Power Point, etc., to create graphical user interface (GUI), access files in memory and other user interactions. The user program simply calls the functions or classes present in the API and pass some variables or parameters, and get the job done.

In the case of this project, I made use of third-party APIs called DirectShow.NET and AForge.NET because there is no managed API for DirectShow. It means that DirectShow is made only for C++ only and not for

.NET. The interface ISampleGrabber is used to grab the individual frames. Because of the complexity involved in implementing DirectShow.NET, we made use of AForge.NET framework library that uses Directshow but exposes functions or methods, classes or interfaces that can be easily used to capture the incoming frames and process them.

The codes that are developed for carrying out various operations in this project are discussed in the form of *Pseudo-Code.* Before we discuss the pseudo code, knowledge of Bitmap Class is a must since it is used. A bitmap is a type of memory organization or image file format used to store digital images. The term bitmap comes from the computer programming terminology, meaning just a map of bits, a spatially mapped array of bits. Bitmap commonly refers to the similar concept of a spatially mapped array of pixels. Raster images in general may be referred to as bitmaps whether synthetic or photographic, in files or in memory. When coming to .NET, a Bitmap is class used to work with images defined by pixel data. The library can be used under the name space in C#.NET as using System.Drawing.Bitmap Figure 2.2 shows a binary bitmapped image. The left half of this diagram shows the bits in the bitmap, and the right half depicts what would show on screen.

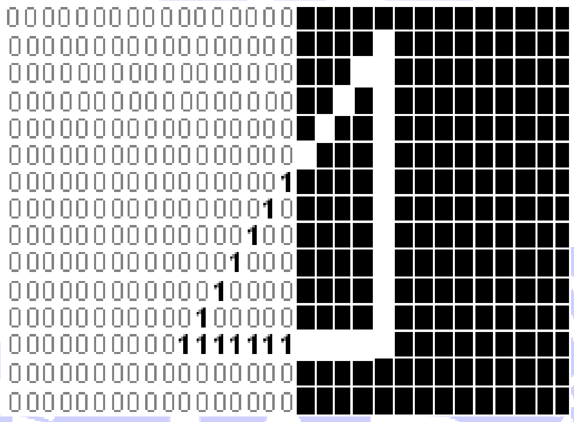


Figure 2.2: A binary bitmapped image

An example of a C# code to load a bitmapped (.bmp) image using the bitmap class is given below.

```
Bitmap image1;
private void Button1_Click(System.Object sender, System.EventArgs e)
{
  try
  {
    // Retrieve the image.
    image1 = new Bitmap(@"C:\Documents and Settings\All Users\"
      + @"Documents\My Music\sathya.bmp", true);
    int x, y;
    // Loop through the images pixels to reset color.
    for(x=0; x<image1.Width; x++)
    {
      for(y=0; y<image1.Height; y++)
      {
        Color pixelColor = image1.GetPixel(x, y);
        Color newColor =
          Color.FromArgb(pixelColor.R, 0, 0);
        image1.SetPixel(x, y, newColor);
      }
    }
    // Set the PictureBox to display the image.
    PictureBox1.Image = image1;

    // Display the pixel format in Label1.
    Label1.Text = "Pixel format:
          "+image1.PixelFormat.ToString();
  }
  catch(ArgumentException)
  {
        MessageBox.Show("There was an error." +
          "Check the path to the image file.");
    }
}
```

What is Pseudo-code?

Pseudo code is a kind of structured English for describing algorithms. It allows the designer to focus on the logic of the algorithm without being distracted by details of language syntax. At the same time, the pseudo code needs to be complete. It describes the entire logic of the algorithm so that implementation becomes a rote mechanical task of translating line by line into source code.

In general, the vocabulary used in the pseudo code should be the vocabulary of the problem domain, not of the implementation domain. Pseudo code is not a rigorous notation and no universal "standard" exists for this representation.

Pseudo-codes for frame grabbing and Image Processing

Before we begin writing the codes, a new project should be created in Visual Studio.NET IDE. Afterwards, include the classes that are used for frame grabbing and image processing, for instance DirectShow. Add the reference to the classes that has been added so that the compiler can identify trace out the library files. Next, two picture boxes should be drawn on the form in the form1 designer tab. Then, open form1.cs file in the editor window and start writing the required code for frame processing.

The following are Pseudo-codes and they should not be treated as actual codes for writing the program in C# in VS.NET editor.

Global Declarations

Declare the integers *pixel*, *DIV=16, sum=0, count=0, pixpos, centroid, diff*

Instantiate an object called *serialport* for RS-232

Initialize the 2-D array *convolution matrix* = {{1,2,1}, //Gaussian

{2,4,2}, //Filter

{1,2,1}} //Coefts

Initialize the 2-D array *dilation matrix* = {{0,255,0},

// Structuring element // {255,255,255}, {0,255,0}}

Form1() //constructor for the form class

Step1: Begin.

Step2: Instantiate object named *videocapture* for acquiring the frames and assign all the frames from video capture device

Step3: Handle Exception by disabling the webcam option in menu if the method videodevice present in the class *videocapture* throws an exception which means that no cameras are detected.

Step4: Assign the portname for the *serialport*

Step5: Assign baudrate for the *serialport*

Step6: Disable parity for *serialport*

Step7: Assign stopbits as one for *serialport*

Step8: Assign databits as 8 for *serialport*

Step9: Disable handshake for *serialport*

Webcam Play event() /*Event invoked when user clicks webcam option */

Step1: Begin

Step2: Stop the other sources to access the buffer

Step3: Assign some desired the frame rate

Step4: Assign some frame size

Step5: Use the start method in the object so that the capturing starts

Step6: Create object that handles a new frame event and process the frame calling the new *frame event handler()*

Step7: End

New frame event handler()

Step1: Begin

Step2: Instantiate two bitmap objects naming img1 and img2

Step3: Copy the entire frame on to the two bitmaps img1 and img2

Step4: Create a clone bitmap object of img2 call it img2cl

/* Image processing starts here */

/* Gaussian Blur Starts */

Step5: Lock the bitmap memory of both img2 and its clone

Step6: Access the memories of both imag2 and its clone using pointers such that the pointers point to a small portion of the memory in the form of a 3x3 matrix

Step7: Extract the colour values using pointers and to process them individually

Step8: Multiply them with corresponding mask coefficients that are stored in the *convolution matrix* 2-dimensional array

Step9: Calculate the sum of all products

Step10: Divide the sum of products with a variable *DIV* and store in variable *pixel*

Step11: If *pixel* > 255 then *pixel* = 255

Step12: If *pixel* < 0 then *pixel* = 0

Step13: Assign the value of pixel to the corresponding middle pixel of the 3x3 matrix of the clone img2cl

Step14: Increment the pointer by 3 (Since B, G & R values are consecutive) and repeat from step 6 to 12 until the last pixel of the bitmap

/* Gaussian Blur Ends */

/* Colour Filter segmentation starts*/

Step15: Access the starting colour pixel memory location of the clone img2cl using pointer after it has been processed using Gaussian filter

Step16: If value at the address pointed by pointer is lying between some range of colour values, then assign its value as 255 else 0

Step17: Increment pointer and repeat step13 until the last pixel of the clone bitmap

/* Colour Filter segmentation ends */

/* Dilation operation starts */

Step18: Create a clone bitmap img2cl2 of img2cl

Step19: Access the starting colour pixel memory locations of both img2cl and img2cl2 bitmaps using pointers such that a 3x3 matrix of locations are pointed

Step20: Compare the pointed pixel locations of the bitmap img2cl corresponding to the elements of the 2-D array *Dilation Matrix*.

Step21: If all are same as that of the structuring element assign the middle pixel of the 3x3 pointer matrix of img2cl2 as 255 else 0

Step22: Increment the pointers and repeat step17 to step20 until the last pixel location

/* Dilation operation ends */

/* Tracking starts */

```
step23: Access the starting pixel memory of img2cl2
Step24: If the pixel value is 255 then assign pixpos = pixel coordinates and increment count
Step25: sum = sum + pixpos
Step26: Increment pointer and repeat step22 to step23 until the last pixel
Step27: centroid = sum/count
Step28: Calculate the difference between the center of the frame or image and centroid
diff = center of image - centroid
Step29: If diff is negative, that is, diff<0 then
open serialport for communication
write charater "R"
close serialport
endif
Step30: If diff>0 then
open serialport for communication
write charater "L"
close serialport
endif
/* Tracking ends */
Step31: Assign picturebox1 image as img1
Step32: Assign picturebox2 image as img2cl2
Step33: unlock the bitmap memory of img2cl2
Step34: End
```

3. Servo Motor Control Using BS2

The computer will be sending the control characters either 'R' or 'L' through the RS-232 serial communication port. The Basic stamp 2 should be ready to receive and read the incoming signal and activate appropriate motor either left motor or the right one, depending on the characters 'L' or 'R'. For this, we use the PBasic command SERIN. The syntax is given as below.

Syntax: **SERIN** *Rpin* {\Fpin}, *Baudmode*, {*Plabel*,} {*Timeout*, *Tlabel*,} [*InputData*]. Its function is to receive asynchronous serial data from RS-232

- ***Rpin*** is a variable/constant/expression* (0 - 16) that specifies the I/O pin through which the serial data will be received. This pin will be set to input mode. On the BS2, BS2e, BS2sx, BS2p, BS2pe, and BS2px, if Rpin is set to 16, the BASIC Stamp uses the dedicated serial-input pin (SIN, physical pin 2), which is normally used by the Stamp Editor during the download process.
- ***Fpin*** is an optional variable/constant/expression (0 - 15) that specifies the I/O pin to indicate flow control status on. This pin will be set to output mode.
- ***Baudmode*** is variable/constant/expression (0 - 7 on the BS1, 0 - 65535 on all other BASIC Stamps) that specifies serial timing and configuration.
- ***Qualifier*** is an optional variable/constant (0 - 255) indicating data that must be received before execution can continue. Multiple qualifiers can be indicated with commas separating them.
- ***Plabel*** is an optional label indicating where the program should go in the event of a parity error. This argument should only be provided if *Baudmode* indicates 7 bits, and even parity.
- ***Timeout*** is an optional variable/constant/expression (0 - 65535) that tells SERIN how long to wait for incoming data. If data does not arrive in time, the program will jump to the address specified by *Tlabel*.

- ***Tlabel*** is an optional label that must be provided along with *Timeout*, indicating where the program should go in the event that data does not arrive within the period specified by *Timeout*.
- ***InputData*** is list of variables and formatters that tells SERIN what to do with incoming data. SERIN can store data in a variable or array, interpret numeric text (decimal, binary, or hex) and store the corresponding value in a variable, wait for a fixed or variable sequence of bytes, or ignore a specified number of bytes. These actions can be combined in any order in the *InputData* list.

One of the most popular forms of communication between electronic devices is serial communication. There are two major types of serial communication; asynchronous and synchronous. The SERIN and SEROUT commands are used to receive and send asynchronous serial data. SERIN can wait for, filter and convert incoming data in powerful ways. SERIN deserves some lengthy discussion, below, since all this power brings some complexity. The term asynchronous means "no clock." More specifically, "asynchronous serial communication" means data is transmitted and received without the use of a separate "clock" wire. Data can be sent using as little as two wires; one for data and one for ground. The PC's serial ports (also called COM ports or RS-232 ports) use asynchronous serial communication. Note: the other kind of serial communication, synchronous, uses at least three wires; one for clock, one for data and one for ground. RS-232 is the electrical specification for the signals that PC serial ports use. Unlike normal logic, where a 5 volts is a logic 1 and 0 volts is logic 0, RS-232 uses -12 volts for logic 1 and +12 volts for logic 0. This specification allows communication over longer wire lengths without amplification.

BASIC Stamp 2 has a line receiver on its SIN pin (Rpin = 16). The SIN pin goes to a PC's serial data-out pin on the DB-9 connector built into BASIC Stamp 2 development board. The connector is wired to allow both programming and run-time serial communication (unless we are using the Stamp 2 Carrier Board which is only designed for programming). For the built-in serial port set the *Rpin* argument to 16 in the SERIN command.

BASIC Stamp 2 can also receive RS-232 data through any of their I/O pins (*Rpin* = 0 - 7 for BS1, *Rpin* = 0 - 15 on all other BASIC Stamps). The I/O pins don't need a line receiver, just a 22 kΩ resistor. The resistor limits current into the I/O pin's built-in clamping diodes, which keep input voltages within a safe range.

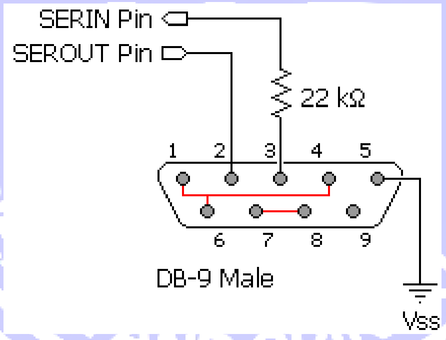


Figure 3.1: RS-232 communication socket

Table 3.1: Pin description of DB-9

DB-9 pin	Description
1	Data Carrier Detect (DCD)
2	Receive Data (RD)
3	Transmit Data (TD)
4	Data Terminal Ready (DTR)
5	Signal Ground (SG)
6	Data Set Ready (DSR)
7	Request To Send (RTS)
8	Clear To Send (CTS)
9	No Connection (NC)

Figure 3.1 shows the pinouts of the two styles of PC serial ports and how to connect them to the BASIC Stamp's I/O pin (the 22 kΩ resister is not needed if connecting to the SIN pin). Though not normally needed, the figure also shows loop back connections that defeat hardware handshaking used by some PC software. It should be noted that PC serial ports are always male connectors. Asynchronous serial communication relies on precise timing. Both the sender and receiver must be set for identical timing, usually expressed in bits per second (bps) called baud. On BS2, SERIN requires a value called *Baudmode* that tells it the important characteristics of the incoming serial data; the bit period, number of data and parity bits, and polarity.

Difference between Bit rate and Baud rate

The difference between the two is complicated and intertwining. They are dependent and inter-related. But the simplest explanation is that a Bit Rate is how many data bits are transmitted per second. A baud Rate is the measurement of the number of times per second a signal in a communications channel changes. Bit rates measure of the number of data bits (that's 0's and 1's) transmitted in one second in a communication channel. A figure of 2400 bits per second means 2400 zeros or ones can be transmitted in one second, hence the abbreviation "bps". Individual characters (for example letters or numbers) which are also referred to as bytes are composed of several bits. A baud rate, by definition, means the number of times a signal in a communications channel changes state or varies.

For example, a 2400 baud rate means that the channel can change states up to 2400 times per second. The term "change state", means that it can change from 0 to 1 or from 1 to 0 up to X (in this case, 2400) times per second. It also refers to the actual state of the connection, such as voltage, frequency or phase level). The main difference of the two is that one change of state can transmit one bit - or slightly more or less than one bit, that depends on the modulation technique used. So the bit bate (bps) and baud rate (baud per second) have this connection: bps = baud per second x the number of bit per baud. The number of bit per baud is determined by the modulation technique.

Here are two examples: When FSK ("Frequency Shift Keying", a transmission technique) is used, each baud transmits one bit; only one change in state is required to send a bit. Thus, the modem's bps rate is equal to the baud rate: When we use a baud rate of 2400, we use a modulation technique called phase modulation that transmits four bits per baud. So: 2400 baud x 4 bits per baud = 9600 bps. Such modems are capable of 9600 bps operation.

Now that BS2 receives and reads the characters from the RS-232 port, it has to send PWM signals to the corresponding servo motor. For this, we have to use the PBasic command PULSOUT. The syntax is as given below.

Syntax: **PULSOUT** *Pin*, *Duration*. Its function is to generate a pulse on *Pin* with a width of *Duration*.

- ***Pin*** is a variable/constant/expression (0 - 15) that specifies the I/O pin to use. This pin will be set to output mode.
- ***Duration*** is a variable/constant/expression (0 - 65535) that specifies the duration of the pulse. The unit of time for *Duration* is described below.

PULSOUT sets *Pin* to output mode, inverts the state of that pin; waits for the specified *Duration*; then inverts the state of the pin again; returning the bit to its original state. The unit of *Duration* is described above. The following example will generate a 100 µs pulse on I/O pin 5:

```
PULSOUT  5, 50   'generate a pulse on pin 5
```

The polarity of the pulse depends on the state of the pin before the command executes. In the example above, if pin 5 was low, PULSOUT would produce a positive pulse. If the pin was high, PULSOUT would produce a negative pulse.

If the pin is an input, the output state bit, won't necessarily match the state of the pin. What happens then? For example: Pin 7 is an input and pulled high by a resistor as shown below. Suppose that pin 7 is low when we execute the instruction:

```
PULSOUT  7, 5    ' generate a pulse on pin 7
```

Figure 3.2 shows the sequence of events on that pin. Initially, pin 7 is high. Its output driver is turned off (because it is in input mode), so the 10 kΩ resistor sets the state on the pin. When PULSOUT executes, it turns on the output.

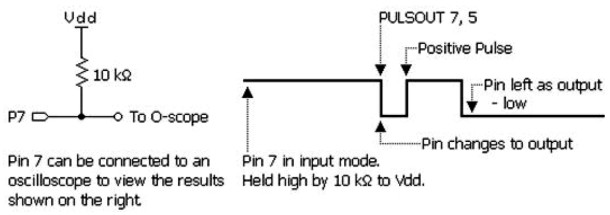

Vdd
10 kΩ
P7
To O-scope
Pin 7 can be connected to an oscilloscope to view the results shown on the right
PULSOUT 7, 5
Positive Pulse
Pin left as output - low
Pin changes to output
Pin 7 in input mode. Held high by 10 kΩ to Vdd.

Figure 3.2: Sequence of events on pin 7

For BS2, the unit for pulse duration is 2µs and the maximum pulse width is 131.07ms. The entire schematic of the Servo control is as shown in figure 3.3. The right servo motor is connected to PIN 13 and left servo is connected to PIN 12 of BS2.

Pseudo-code for servo control

The following Pseudo-code is meant for programming in PBasic programming language.

```
Step1: Start
Step2: Declare constants CON T2400 = 396, T9600 = 84, Inverted = $4000, Baud9600 = T900 + Inverted
Step3: Declare variables VAR command = byte
Step4: Read the RS-232 com port at PIN 16 and store the ASCII character in the variable command
Step5: If command = “R” then
Send PWM signals to PIN 12 which connects the left servo motor
endif
```

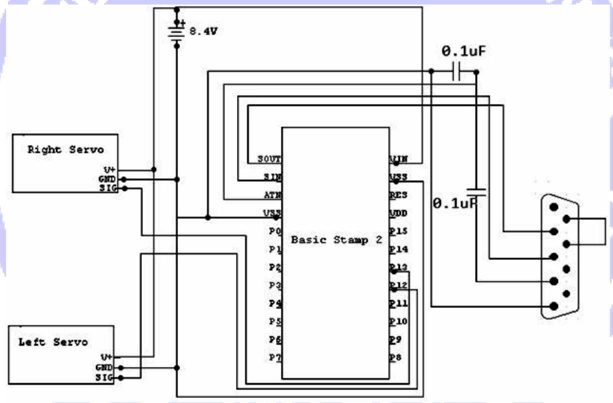


Figure 3.3: Schematic of the Servo control

```
Step6: If command = “L” then
Send PWM signals to PIN 13 which connects the right servo motor
endif
Step7: Repeat step3 to step5 forever until RST pin goes low
Step8: End
```

4. Results, Conclusions and Future Perspectives

4.1 Results

The results obtained while operating the robot are presented in this section in the form of digital images frozen from video streams. Each video frame is self-explicable with its caption.

The video stream is obtained from the web camera of the robot. Using Gaussian low pass filter, the image is blurred, which could be clearly seen in the figure 4.1.1. Note that the sharp edges are lost in the processed frame. The low pass filter could be applied to recorded video stream too. Figure 4.1.2 shows a sample recorded video stream frame and its low pass filtered frame. The sharp edges could be seen to be lost in the processed frame.

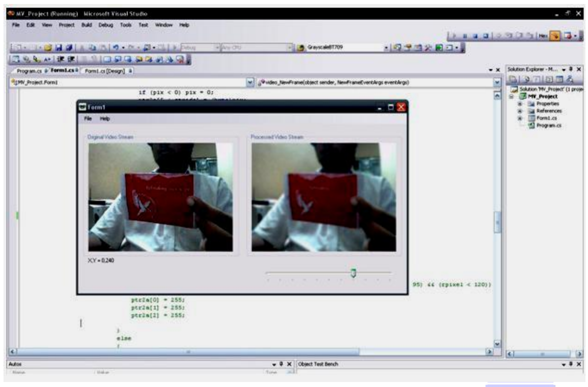
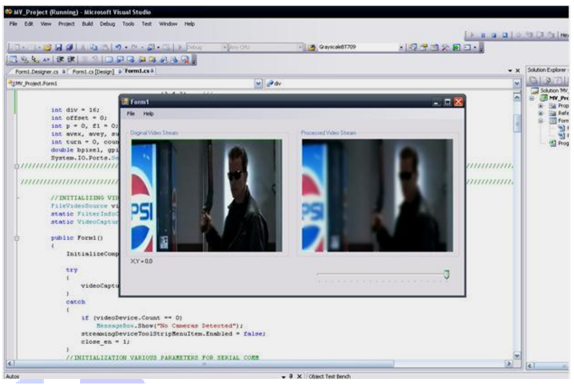


Figure 4.1.1: Gaussian Blur Filter applied to a live video stream



```
int div = 16;
int offset = 0;
int p = 0, f1 = 0;
int avex, avey, su
int turn = 0, coun
double bpixel, gpi
System.IO.Ports.Se
//////////////////////////

//////////////////////////

//INITIALIZING VI
FileVideoSource vi
static FilterInfoC
static VideoCaptur

public Form1()
{
    InitializeComp

    try
    {
        videoCaptu
    }
    catch
    {
        if (videoDevice.Count == 0)
            MessageBox.Show("No Cameras Detected");
        streamingDeviceToolStripMenuItem.Enabled = false;
        close_en = 1;
    }
    //INITIALIZATION VARIOUS PARAMETERS FOR SERIAL COMM
```

Figure 4.1.2: Gaussian Blur Filter applied to a recorded video stream

Our programs run only on frames stored in the video buffer. In order to process AVI files, one has to uncompress the AVI file, sequentially store frames in a buffer, process the buffered frame, and sequentially display the processed frames on the monitor.

Input specifications

Type of the video: AVI

Frame Rate: 25

Colour Spec: 24-bit RGB

Based on colour information (red), the object is isolated after low pass filtering the video frames twice and then thresholding it with a priori colour value (RGB) and binarization.

Note that the isolated and binarized video frame image is highly sparsed and disconnected at various places. In such a situation, it is very difficult to fix the tracking point (say, centroid) in the image.

A feasible solution to this problem is to dilate the image using morphological dilation operator with the help of a suitable structuring element. Bigger the size of the structuring element, better the dilation so that most of the disconnected regions would get connected. The only disadvantage here is that the speed of processing the video would reduce considerably. Figure 4.1.4 shows the dilated version of the image shown in figure 4.1.3.

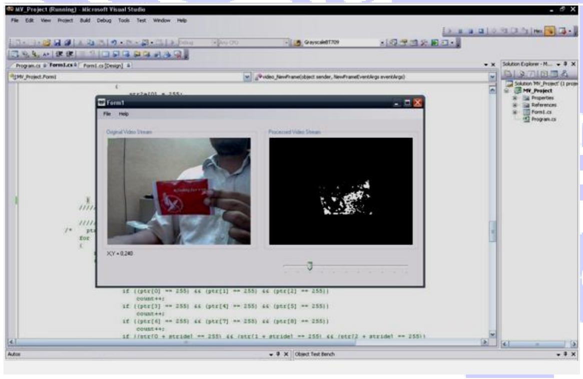


Figure 4.1.3: Isolation of object based on colour information

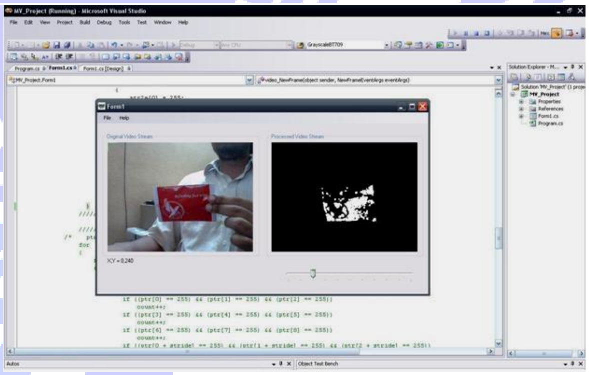


Figure 4.1.4: Dilation of disconnected regions

Contouring is yet another operation used to solve different pattern recognition problems. Figure 4.1.5 shows a recorded video frame and its contour map. Laplacian operator is used to obtain the contour map.

The Laplacian of a 2-D function a[i, j] is a second order derivative defined as

$$\nabla^2 a = \frac{\partial^2 a + \partial^2 a}{\partial i^2 + \partial j^2}$$

Five Laplacian masks have been shown below for ready reference and use.

$$\begin{array}{ccccc}
\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} &
\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} &
\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} &
\begin{bmatrix} 1 & 1 & 1 \\ 1 & -7 & 1 \\ 1 & 1 & 1 \end{bmatrix} &
\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \\
\text{Laplacian Mask \# 1} & \text{Laplacian Mask \# 2} & \text{Laplacian Mask \# 3} & \text{Laplacian Mask \# 4} & \text{Laplacian Mask \# 5}
\end{array}$$

Another way of obtaining edge map is by applying Sobel operator to the video frame image. Sobel operators are gradient operators. Gradient of a digital image a[i, j] at location [i,j] is the vector ∇a is the transpose of the row vector [Gx,Gy] which is the transpose of the row vector [(∂a/∂x) (∂a/∂y)]. The edge of an image feature corresponds to the magnitude of this vector which is given by $[G_x^2 - G_y^2]^{1/2}$. Six different masks are shown below for reference and use.

$$\begin{array}{cccccc}
\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} &
\begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & -2 \end{bmatrix} &
\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & 2 \\ 1 & 0 & -1 \end{bmatrix} &
\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} &
\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} &
\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 2 \end{bmatrix} \\
\text{Sobel Mask \# 1} & \text{Sobel Mask \# 2} & \text{Sobel Mask \# 3} & \text{Sobel Mask \# 4} & \text{Sobel Mask \# 5} & \text{Sobel Mask \# 6}
\end{array}$$

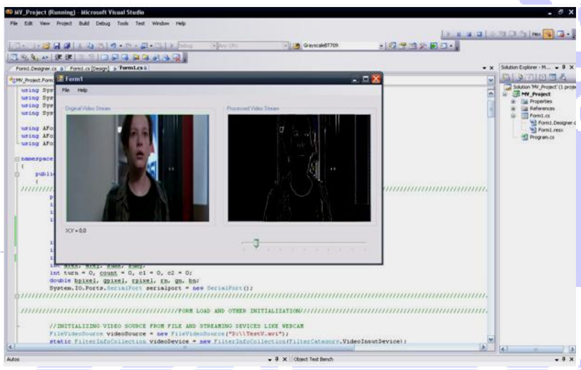


Figure 4.1.5: Video frame with its edge map obtained using Lapalcian operator

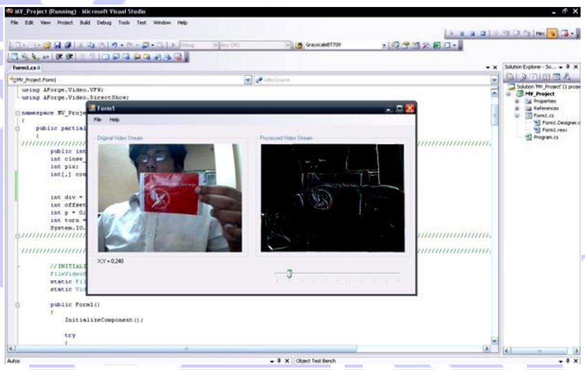


Figure 4.1.6: Video frame with its edge map Obtained using Sobel operator

Figure 4.1.6 shows a frame of video from web camera and its edge map obtained using Sobel operator. After isolating the desired object, one has to fix up the track point in the image.

Many techniques are being in use to obtain the centroid of an object image. Iterated thinning is one such tool by which one can fix up the centroid.

Given an image one can apply thinning algorithm to it and remove its boundary by one pixel. The thinning process repeatedly applied to the image till one ends up in a situation that there is no boundary left. The remaining point is called the centroid of the object in an image. Figure 4.1.7 shows a live video from the web camera where in a colour object is shown before the camera and the robot identifies its centroid frame by frame. A sequential identification of centroids leads to what we call as tracking.

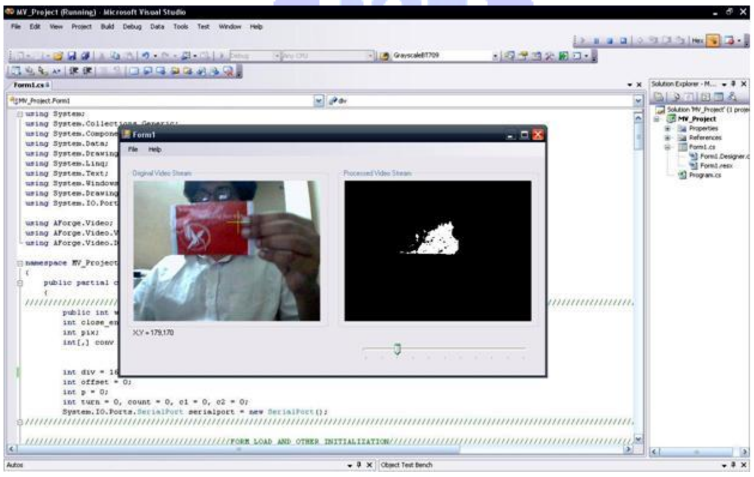


Figure 4.1.7: Cross-hairs tracking the red object (refer to the '+' mark on the red coloured object)

In what follows, I provide the pictures of the entire real time machine vision-based tracking robot developed as the main product of this project as a proof of concept.

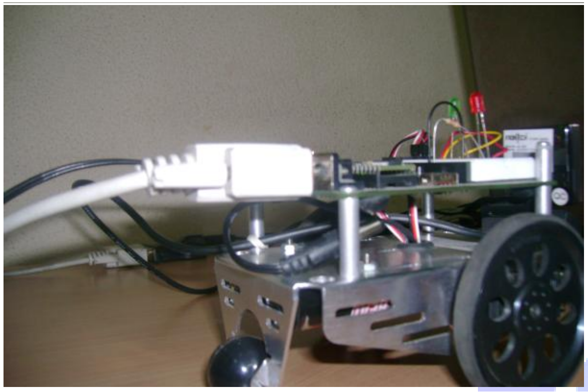


Figure 4.1.8

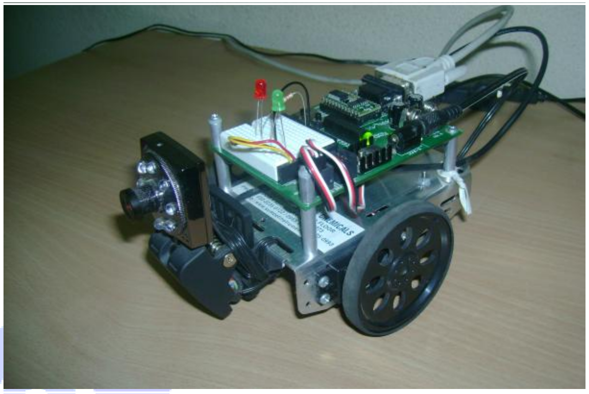


Figure 4.1.9

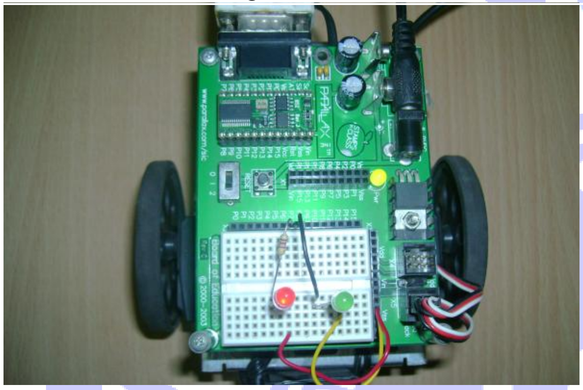


Figure 4.1.10

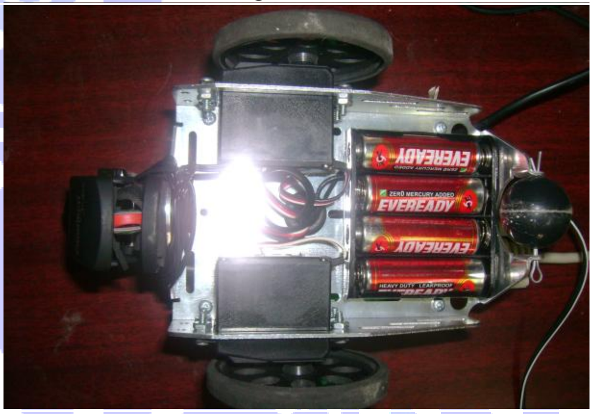


Figure 4.1.11

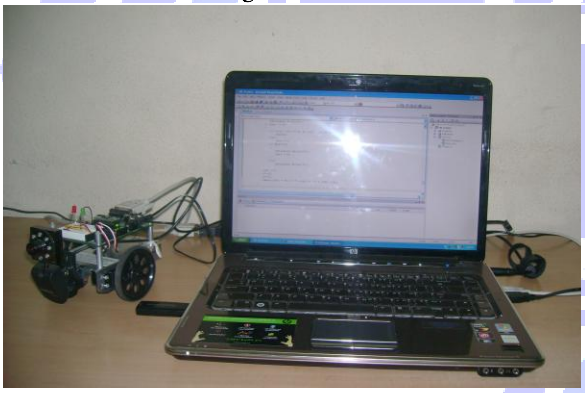


Figure 4.1.12

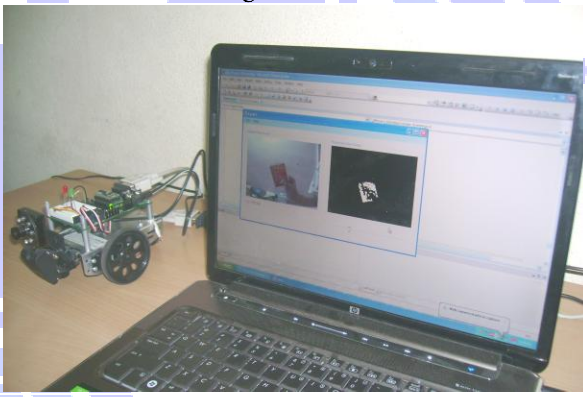


Figure 4.1.13

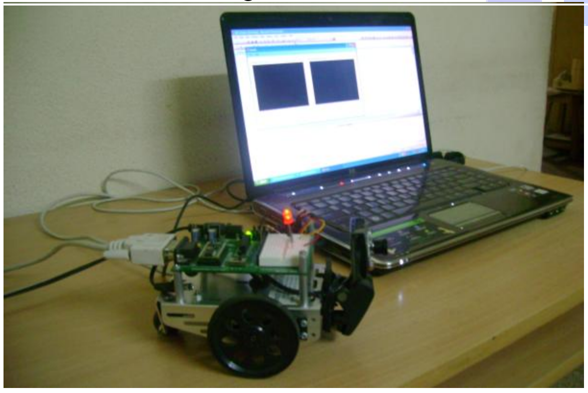


Figure 4.1.14

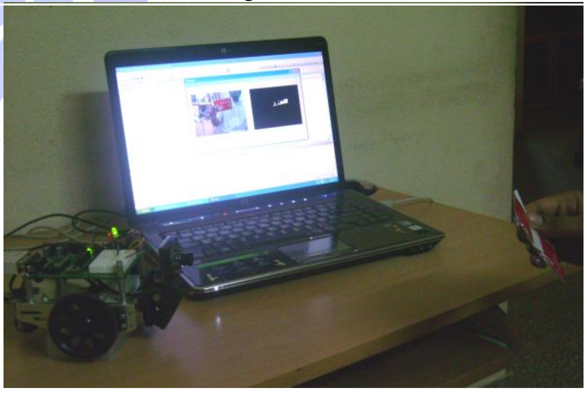


Figure 4.1.15

Legend:

Figure	Description
4.1.8	Rear view of the robot showing RS-232 connection.
4.1.9	Side view of the robot fitted with a web camera
4.1.10	Top view of the robot showing the BS2 microcontroller and the Board of Education
4.1.11	Bottom view of the robot showing two Parallax Servo motors attached to the wheels
4.1.12	Robot connected to HP Pavillion DV-5 1106AX laptop through serial port
4.1.13	Laptop screen shows the object tracking video frame
4.1.14	Red LED on the robot glows indicating that the object is lost and hence no tracking
4.1.15	Green LED on the robot glows indicating that the object has been found and the tracking has started

4.2 Conclusions

Boe-Bot Robot Kit was purchased from its manufacturer Parallax. This robot consists of Basic Stamp 2 microcontroller, Board of Education, Robot's aluminium body, two Parallax servo motors, four 1.5V AA sized battery and a RS232 serial cable. A web camera was installed in the robot for video capturing purposes. The camera was connected to a computer for processing the video frames.

"Vision is the most advanced of our senses, so it is not surprising that images play the single most important role in human perception". - an excerpt from book of digital image processing by Gonzalez and Woods. So, it can be seen that vision is the most important sensor which enables us to interact with the environment better than any other sensory perception. Hence it should make sense why machine vision is the most powerful tool for robot guidance. Robots have evolved from a simple mechanical clock to complex humanoids. But without proper guidance based on sensors, there will be no difference between a robot and a toy. Machine vision has proven to be an able guiding method which involves various parameters.

Applications of machine vision are many. Some of useful applications are as given below.

- Machine vision-based missile guidance can be a best alternative for radar or laser-based guidance. The reason why machine vision is so powerful in guidance is that it we have got many characteristics in our hand to work with. For instance, characteristics such as colour, pattern, edge, etc, can be used to scrutinize friendly and enemy targets, which cannot be done in conventional radar and laser guided missiles.
- In industries where there is a requirement for inspection of various things which are routine and which require quantitative analysis rather than qualitative.
- In security and surveillance where there is a need to recognize, track, and observe subjects.
- Machine vision-based robots can be sent to places where there is a threat for human survival and places where the robot should completely act on its own as the human monitoring and control is out of scope. The best example can be space exploration in which there will be a huge communication gap between the machine that has been sent to outer space and human beings on earth.

4.3 Future perspectives

There is a long way for to go for machine vision-based robot to exactly imitate human vision perception. Continuing efforts such as this project on machine vision field, can possibly accelerate its evolution and can make machine vision robots soon attain capabilities as close as possible to human vision. Although some machine vision based automated machine prototypes have been made available, they are not completely reliable and fault tolerant. Although the applications are many, machine vision hasn't been completely realized till date due to many technical limitations. As with many other technologies, there are some future perspectives for it with an assumption that there will be no technological limitations in the future. As machine vision-based systems are quantitative rather than qualitative, they can replace human beings in places where precision and hygiene are top priorities. Such a place can be a semiconductor industry. With high resolution cameras, the precision of the machine vision-based systems can be increased. Artificial intelligence which is mostly based on probability and statistics, can give a big boost to machine vision systems such as a machine vision-based missile. The person who launches the missile can just fire and forget about it.

About the Author

Sathya Govindarajan obtained B.Tech (ECE) degree from Jawaharlal Nehru Technological University, Hyderabad, India and MS degree from New York Institute of Technology, Old Westbury, New York, USA. He worked in DARPA sponsored project on continuous authentication – Key Stroke and Touch, in NYiT. He was also involved in the application development for Continuous Facial Authentication. At present, he is working as a Senior Lead in Automatic Data Processing Inc. (ADP), New Jersey, USA. His areas of interest include Data Science, Signal and Image Processing, Robotics and Automation, Advanced Communication Systems, Computer Animation and Graphics.

A Humble Dedication

This paper is dedicated to our Guru Dr. E. G. Rajan who taught the fundamentals of Signal and Image Processing and who motivated me to look into the possibilities of developing programming tools to process 2-D and 3-D images and digital video frames. I thank those peer review committee members for their very valuable criticism and support in bringing out this paper to this form.

G. Sathya

Congratulations Mr. Sathya
Organizing Committee, IAPIC 2025

Guiding robots with machine vision enables flexible manufacturing and production lines to readily accommodate product changes. In addition to locating parts for pick-and-place or guiding a robot to assemble components, machine vision can also inspect, measure, and read 1D and 2D barcodes as products are being handled or assembled. It also eliminates costly precision fixturing, prevents accidental robot collisions, and processes various part types without a tooling changeover.

Courtesy: https://www.cognex.com/industries/vision-guided-robotics