CHAPTER 1

INTRODUCTION

1.1 MOTIVE

Auto assembly is one of the most common issues in the production lines of any industry. Many assembly lines use autonomous robots guided by the lines along their path. Changing the layout of machinery with respect to newer designs is cumbersome in such systems. The level of congestion that industry tolerates is a rational choice between the costs of improving the transportation system (in infrastructure or management) and the benefits of quicker travel. So we need a better system to speed up and automate the process to a new better level.

When we consider todays automated assembly lines of many major car manufacturing companies or even Mother board assembly lines for computers, we find significant size autonomous robotic arms operating on chassis (in case of cars) and through hole components (in case of mother board assembly) but still exterior parts (in car manufacturing) and certain functionally dilated components are manually assembled by human labor. This is because of a evident limitations of the machines mentioned by the manufacturers. This project uses image processing to identify the position of object and guides the robot to place it in the appropriate destination regardless of the layout structure of production line viz. Process type layout and Product type layout.

The installation of this system helps in reduction of space occupied by belt and pulley mechanisms used to transfer Integrated Circuits to the site of motherboard assembly.

1.2 OBJECTIVES

Development of prototype image processing based robotic system can definitely prove to be a feasible technology. The future of assembly system is emphatically all about automated transport systems and vehicles that can drive themselves around all available room reducing space occupied by mechanical systems for transportation especially belt and pulley drive mechanisms. The main objective of the project is to develop a system to aid the existing automated assembly lines to work much more independently with much less human intervention.

1.3 THESIS ORGANIZATION

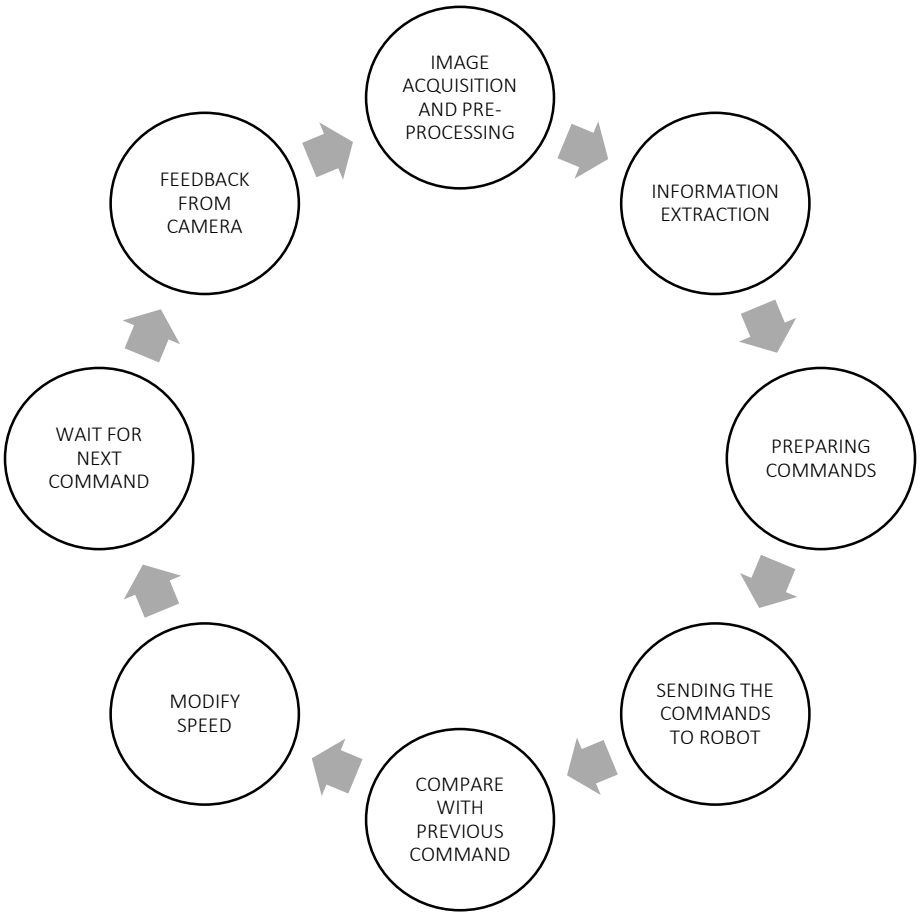


Figure 1.1 Block diagram of the entire system

In this chapter we have discussed about the overview of the project and the main objective behind using this project.

Chapter 2 deals with real time embedded systems its architecture and how we divided the project. Here the details of arrival to the employed complete embedded system for the purpose of project is described with each of the blocks of the system discussed in detail in further chapters.

The very first block of the entire system, image acquisition and preprocessing is discussed in the third chapter. Not all images acquired can be directly operated on to get the required information we desire. For the process of extraction of the required information from the images they must be converted to a suitable form. The different ways of representing an image (Vector and Raster) and different color space component representations of digital color images are dealt in this chapter. Such operations for preprocessing thus include colorspace conversion, compression and expansion of images.

The two blocks precisely, processing and thus extracting information from the acquired images after preprocessing along with the generation of commands for functioning of the actuators in robot are discussed in chapter 4. This block involves certain techniques such as smoothening, contrast adjustment, gray - level transformation, gray to binary conversion or thresholding, algebraic operations, morphological processing through which presence and orientation of different objects in workspace with respect to destination are determined based on which commands for robot are also generated.

Chapter 5 speaks about one of the most important topics of the project, communication of central system with Relocator robot which evidently covers the architecture of the Relocator too. The commands generated by central system as described in chapter 4

are to be communicated to the robot for actuation. The way received information is perceived by the robot and all other parameters will also be discussed in chapter 5.

Chapter 6 deals with the description of the Hardware used in this project i.e., complete description of all the components used and specifications of them. This chapter does not include complete datasheet of any of these components but lists only their features and of them the features being made use of in the chapter are discussed.

Chapter 7 deals with the description of software's we have used and steps to use them to make the project.

Chapter 8 deals with the results of all the work related to the project and conclusion of how the present system can be implemented in real-time considering a motherboard manufacturing assembly line.

Chapter 9 deals with the Scope for future work.

Bibliography deals with text books referred along with the websites which provided us useful content for the project.

Appendix of the project deals with the algorithm related to the project, circuitry and circuit board used, its layout, USB programmer and USB to TTL converter.

1.4 CONCLUSION

After providing a brief description on this project and what this thesis project will hopefully achieve. We also hope that this research can come up with a system that can help develop an automated system to reduce time to assemble the product.

CHAPTER 2

ARCHITECTURE

2.1 INTRODUCTION

The working of the DIP based Autonomous Locomotive system is divided into 4 stages, they are

1. Image acquisition and Pre-Processing
2. Identification of object, destination, robot and its orientation
3. Generation of commands for robot
4. Communication between workstation and Robot - encoding and decoding of commands in respective subsystems and hence movement of robot

Stage 1:

The moment Robot is switched on, information from overhead camera is taken and processed by Workstation (here, PC) and preprocessed to a form suitable for processing in later stages.

The moment object is placed in the arena, images are acquired in YCrCb form. In this stage image is converted to RGB form. The three color space components are extracted from the resultant RGB image and each of the component images are pre-processed to identify different regions it has been split into.

Stage 2:

The properties of split regions are extracted from the color space components and presence of object and if so, its destination are obtained

Stage 3:

Depending on position of the object, destination, robot and its orientation Matlab prepares the commands (here single character commands are used) to be sent to robot through serial communication – moving in 2 Dimensional ground plane.

Stage 4:

This is the final stage of the process where commands generated by Matlab are sent into air by a transmitter module. The Robot with a receiver module after receiving those commands decodes the information regarding actuation from the serial data and performs accordingly.

Since the commands include waking and sleeping of robot for effective power management, we need a correct feedback system to determine appropriate commands which is taken care of along with any deviations or errors by the very first stage operations.

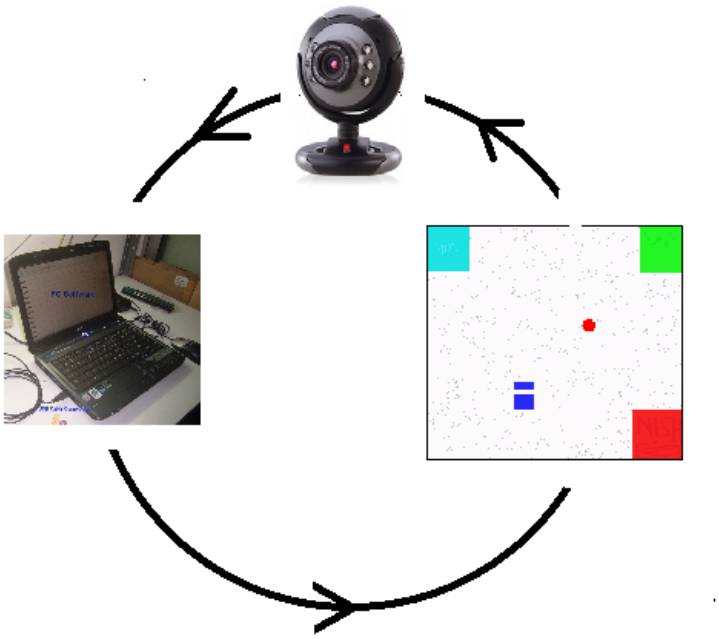


Fig 2.1 block diagram

From the above figure (Fig 2.1) it is quite clear that the central system monitors assembly procedure and commands robot near the position where there is an unused object for relocation.

We have divided the system into to two main divisions

- Autonomous Locomotive System(Robot)
- Central system

2.2 ROBOT

Here the Robot is just like a vehicle which moves from its position to object based on whether the object is in use or to be placed in its deck. In the proto type we have used a simple robot that uses serial communication to receive commands from central station.

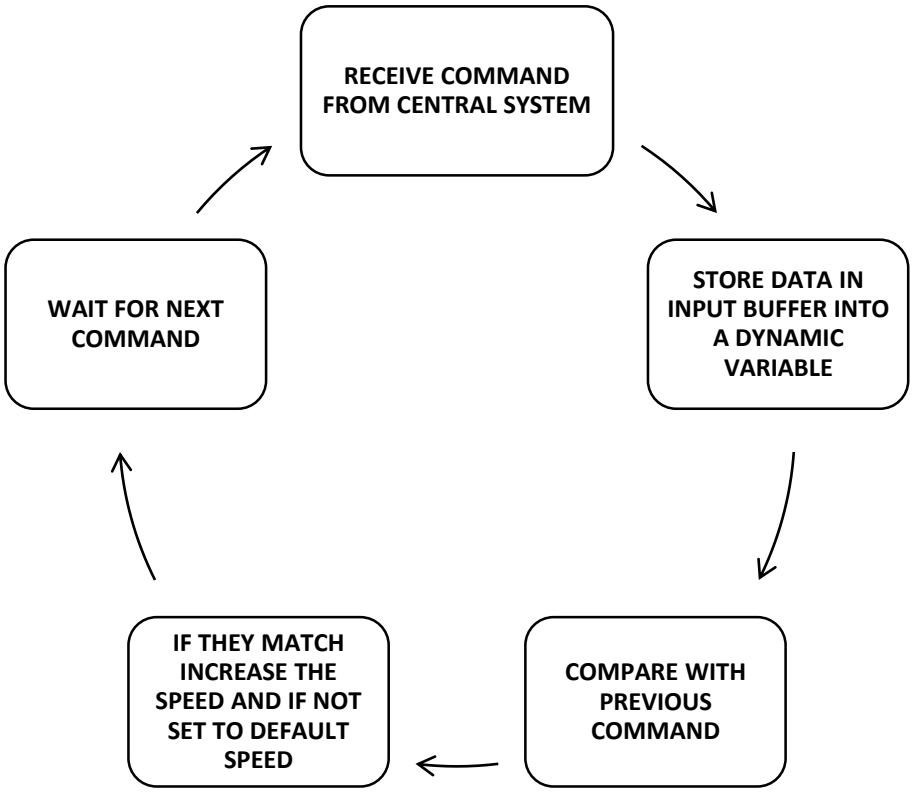


Fig 2.2 block diagram of Robot

The brain of the Robot is micro controller where it does all the computations to decode the information from the workstation and move accordingly.

Here input commands to the microcontroller is taken from RF receiver module through USART interface and this information manipulated based on the code in the controller and the logic for running the motors is sent through the output pins of the microcontroller. By this way the motors are controlled and operated in accordance to received commands. RF transceiver module allows the two way communication between both robot and central station and always will be under control of the central system.

2.2.1 ARENA

Arena has a dynamically fixed or predefined destination used for the project where the Robots move over available space without any predefined paths by the way they have been programmed.

The arena contains predefined destinations of different predefined colors. The bots have to be moved on the white space of arena towards the object and relocate it at corresponding destination.

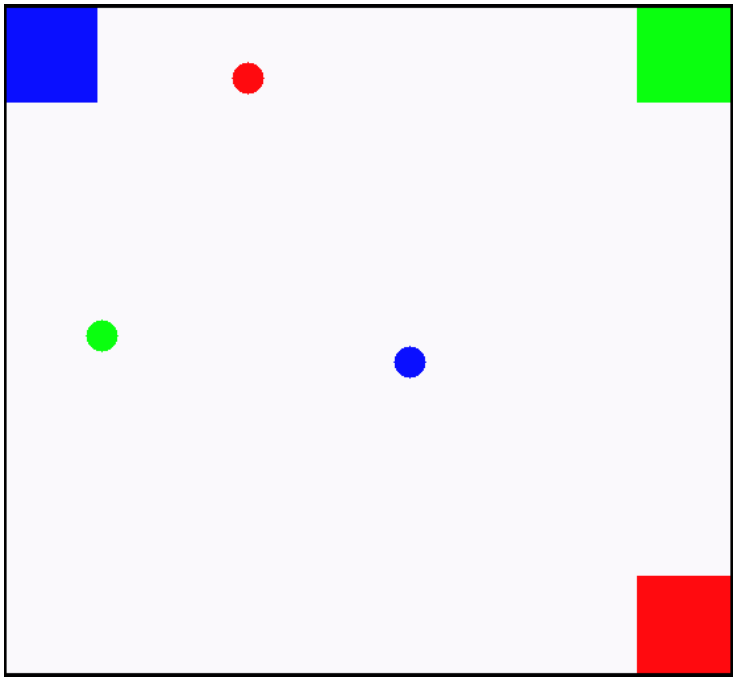


Fig 2.4 structure of ARENA

The destination is determined by central system based on the color of the object. The robot is tagged with a colour different from the ones used to represent the objects and their destinations.

2.3 CENTRAL SYSTEM

This can also be called as server of the entire Production/Assembly system, where the central system will be having the full authority on all the assembly equipment, Robots and these Robots will be functioning on the commands given by the central system. Initially central system will be keeping track of objects (equipment or material) present at the work place unused and sends commands to robot to move to the site and relocate them at their usual positions. By this way central system acts as a server and takes hold of the entire work place and avoids unnecessary accumulation of objects in the arena by providing different paths to different robots during busy hours.

The block diagram of central system represented in our project consist of a Personal Computer with image Processing software installed (here Matlab), a USB to TTL converter or a UART bridge circuit and transceiver module where the transceiver will be doing one way communication between the Robot unit and the data is transmitted to Robot through this device.

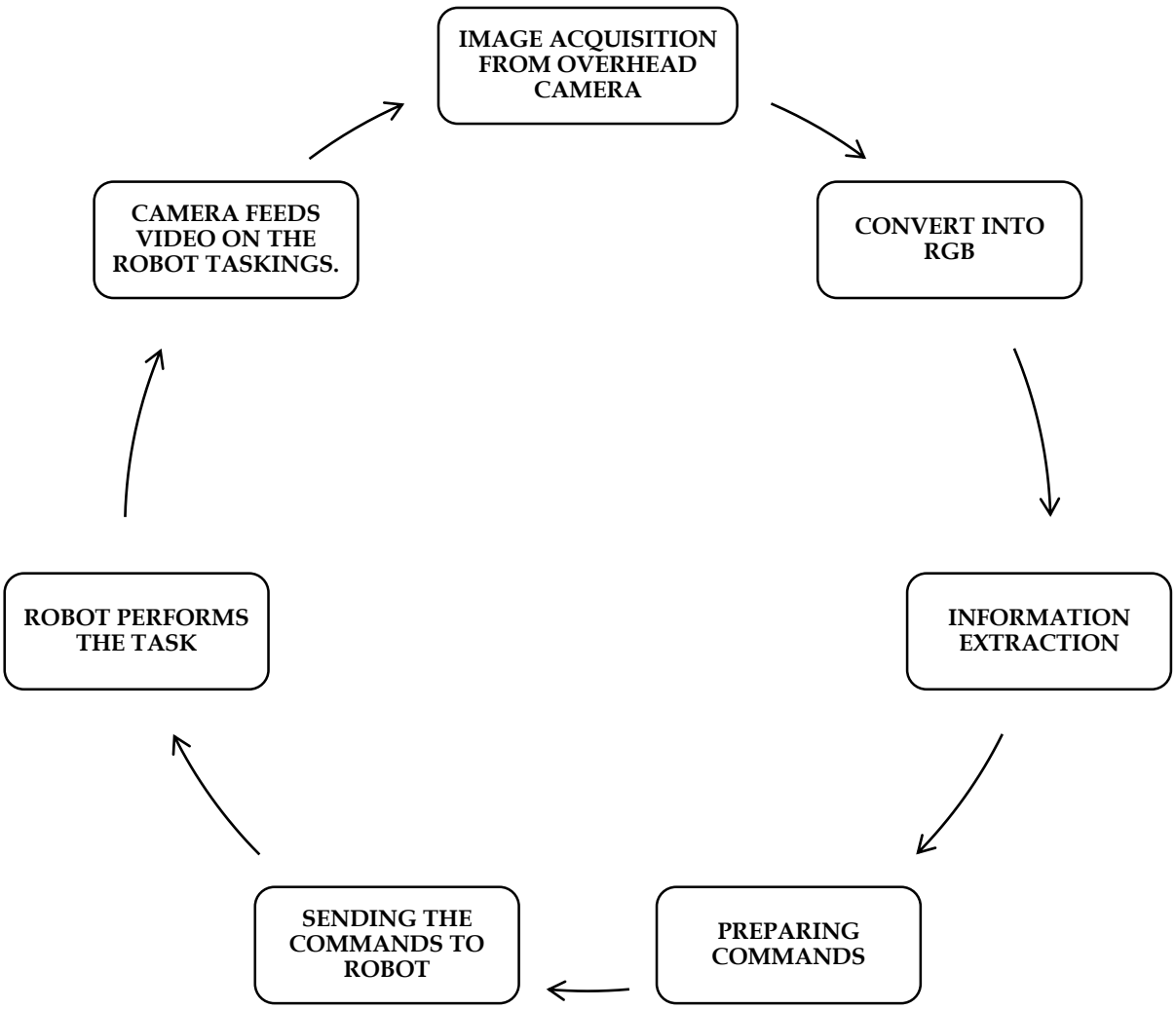


Fig 2.3 block diagram of central system

2.4 CONCLUSION

The architecture of project and an over view of all the building blocks of the project is discussed.

CHAPTER 3

IMAGE ACQUISITION AND PRE-PROCESSING

3.1 INTRODUCTION

Image acquisition could be as simple as being given an image that is already in a form and suitable for processing. In any real-time applications one needs to transform the captured image(s) into a form, other than in which it is captured, suitable for processing.

3.2 IMAGE

An image may be defined as a two-dimensional function, f(x, y), where x and y are spatial (plane) coordinates, and the amplitude of 'f' at any pair of coordinates (x, y) is called the intensity or gray level of the image at that point. When x, y, and the intensity values of f are all finite, discrete quantities, we call the image a digital image.

Cameras are used to acquire images from real-world scenes. There are different kinds of cameras depending on properties of images it captures. They are,

1. Analogue Camera
2. Digital Camera
    a. CCD based camera
    b. CMOS based camera

Figure 3.1 CMOS based web camera

Analogue cameras require grabbing card or TV tuner card to interface with a Personal Computer. The field of digital image processing refers to processing digital images by means of a digital computer. Computers cannot handle continuous images but only arrays of digital numbers. It is so required that all the images to be represented as the digital numerals. A digital image can be represented in either of two standard forms viz. Vector and Raster.

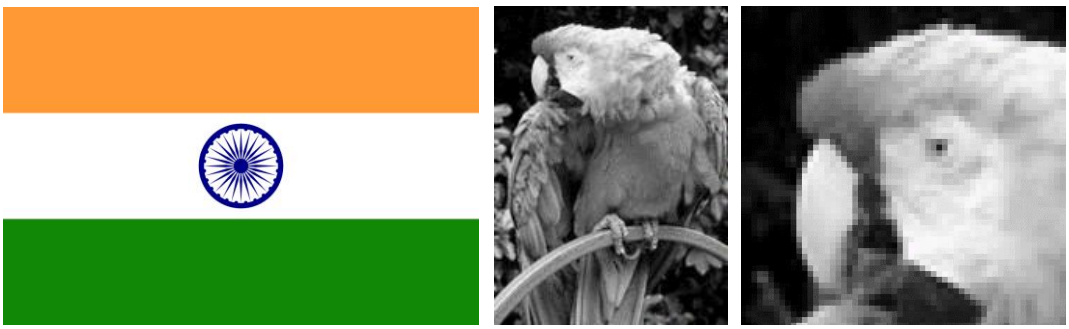


Figure 3.2 (a) Vector image. (b) and (c) Raster image.

Vector images stores curve information and graphic primitives such as straight lines, polygons and points. Raster images are made up of finely and uniformly divided point segments called pixels. That is raster images are represented as 2 dimensional matrix of numeral data. Most of the commonly available hardware and software in this digital

world supports Raster images, for example a laptop monitor. Matlab is one such software.

3.3 IMAGE ACQUISITION

Image Acquisition is the creation of digital image, typically from a physical scene. As mentioned in the previous chapter the project uses a single camera system for the purpose of image processing. A camera is mounted on celling such that the camera feed covers the whole region, a surface with white background and coloured patches at distinguishable positions. The robot, colour patches and objects are all supposed to be covered within the video feed.

Image processing is quite a vast field to deal with. We can identify colours, intensity, edges, texture or pattern in an image. In this project, we would be restricting ourselves to detecting colours (using RGB values) only.

Since present day computers viz. laptops are provided with integrated web camera accessing an external camera involves identification. Image acquisition app of Matlab provides all information about image acquisition hardware (device id, supported formats, supported resolution, adaptor to interface the device …) to select the device of our concern to be used for the project.

A video object is to be created in Matlab with the properties extracted from IMAQTOOL. The video feed is dynamically associated with this video object and is previewed for user interface. Matlab code operates on the most recently acquired video frame by obtaining snapshot of feed video.

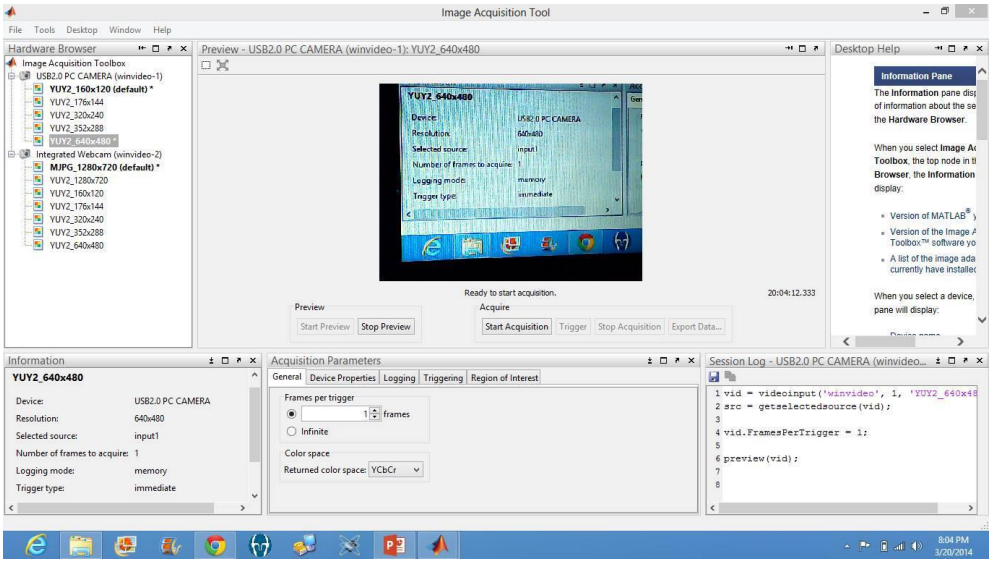


Figure 3.3 image acquisition tool app of MATLAB

3.4 IMAGE PREPROCESSING

Image frame obtained from the hardware used in this project is of colour space YUV, a digital equivalent of YCrCb. But the processing to be dealt in the next chapter deals with images of RGB colour space.

There are algorithms to convert the images between different colour spaces. Matlab provides a way to obtain images from device through preview pane directly in required format. By adjusting colorspace of image to be returned by video object out of the preview pane to required format.

$$B = 1.164(Y - 16) + 2.018(U - 128)$$

$$G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128)$$

$$R = 1.164(Y - 16) + 1.596(V - 128)$$

Where Y is the luminance component, U and V are chrominance components of the acquired image. Also R, G and B are two dimensional matrices denoting intensities of red blue and green components of the image.

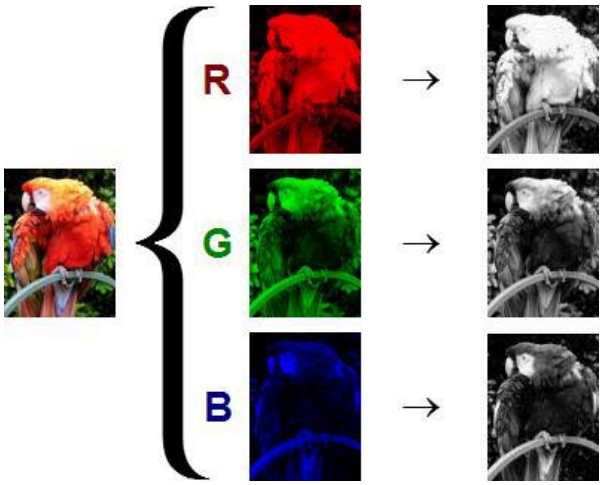


Figure 3.4 RGB colour space decomposition

The other operation to be performed on the image before processing is cropping which eliminates the unwanted and the erroneous data around the borders of the arena

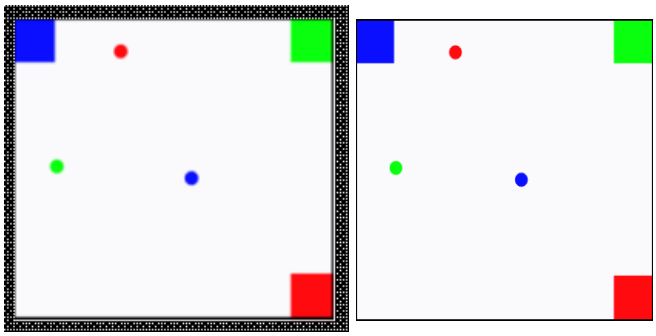


Figure 3.5 Image cropping

3.5 CONCLUSION

The hardware required and adjustment of parameters required for image acquisition along with pre-processing of the acquired images to be transformed into a format required for next stages of processing have been described.

CHAPTER 4

INFORMATION EXTRACTION

4.1 INTRODUCTION

Before one starts processing the images acquired from the hardware device, it is advisable to process the estimated images and apply the same on the real time images. In this chapter all the operations are performed on many estimated images depicting the arena, objects and robot. Of all those operations a few suitable operations are chosen to be applied on the real time images based on the external conditions viz. ambient light, focussing of light and shadows…

4.2 IMAGE PROESSING

In the previous chapter it was shown how images are acquired in the form suitable for processing. Images are obtained in RGB colorspace by the start of the processing. Operations such as DCT, FFT… are performed on the acquired images to obtain the parameters such as frequency, contrast… but there are also certain operations to transform images in space domain such as graylevel binary negative transformations.

Following are the list of operations likely to be imposed on the images,

1. Smoothening
2. Contrast adjustment
3. Graylevel transformation
4. Binary transformation or thresholding

5. Algebraic operations
    a. Adding
    b. Subtracting
6. Morphological processing
    a. Morphological opening and closing of images
    b. Clearing ridges at the borders / morphological filling operation (this is different from smoothening)
    c. Identifying independent regions
    d. Labelling of different regions
    e. Finding area and centroid of the identified regions

4.2.1 TEST IMAGES

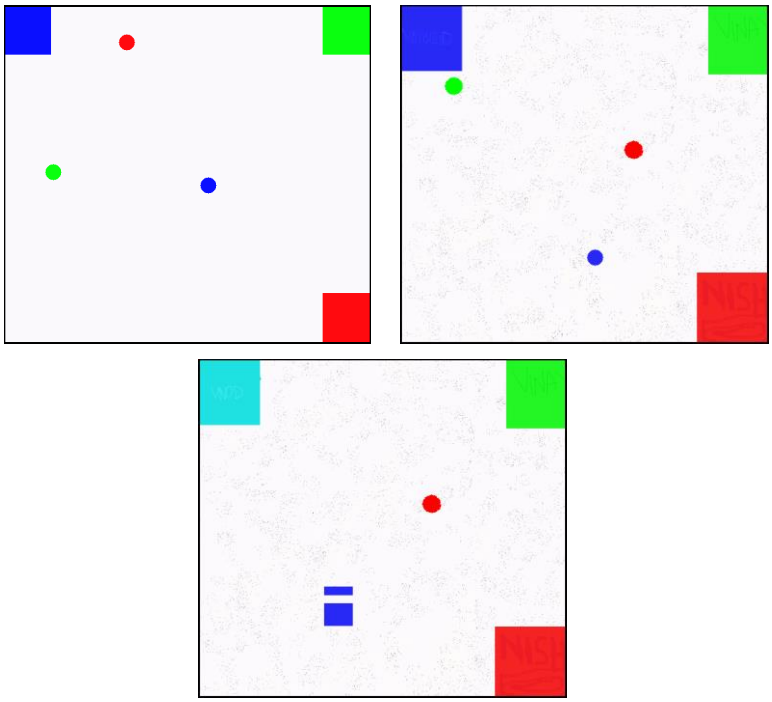


Figure 4.1 Test images.

The images chosen for testing the project are so designed that the colours at corners represent destinations circles represent the objects and that of robot is chosen in different colour with two patches of different area to distinguish between head and tail or determining the direction it points. Few images are designed to incorporate noise patterns also. Images mentioned in fig.4.1 are not the only test images used and further more images and their use are described in later part of the chapter.

4.2.2 INTENSITY IMAGES OF EACH COLOR PLANE

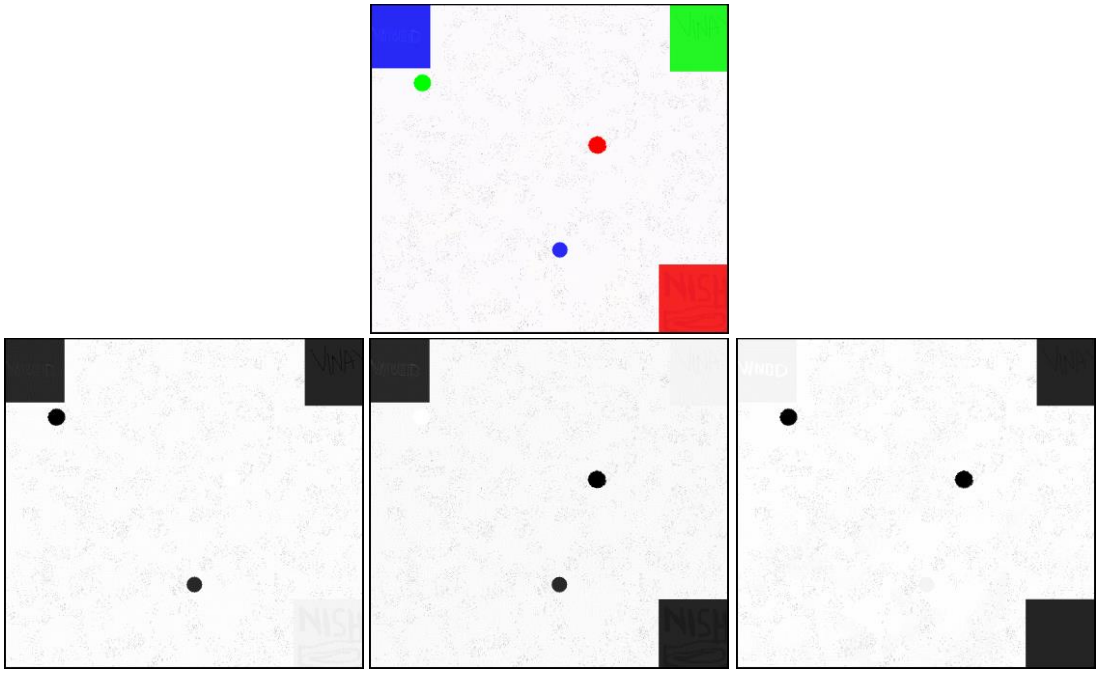


Figure 4.2 Red Green and Blue colour space components

Firstly all the three colour space components are squeezed out of the image and are represented as three different and independent grayscale images. Figure 4.2 gives a quite broader idea about the squeezing of component. Darker the pixel is, lesser is the intensity of the colorspace component at that point.

Here squeezed component images are 2 dimensional grayscale images with each pixel attaining a value in the interval [0,256] (by default it is an unsigned 8 bit integer). Value 0 denotes there is no share of the colour component towards the intensity of the corresponding colour image at that position where any other value denotes presence and amount it shares as fraction of total intensity depending on share of other colour space components also.

4.2.3 THRESHOLDING

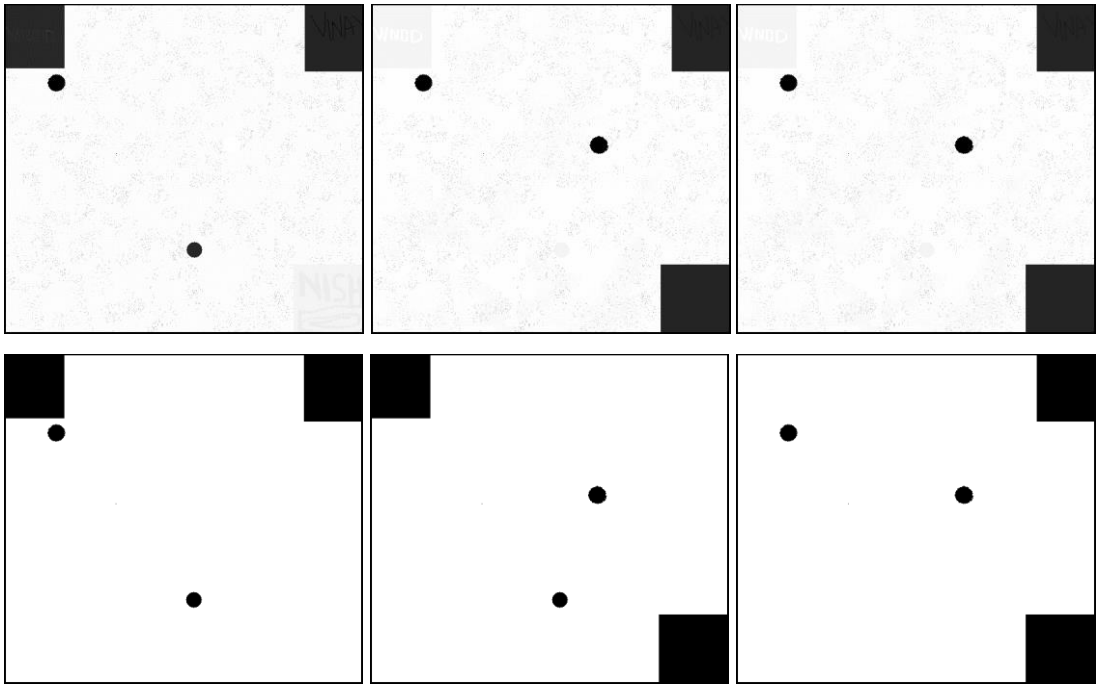


Figure 4.3 Binary images threshold out of colorspace components of test image

4.2.4 SEGMENTATION AND THRESHOLDING

$$Red\_only = binary\_red - binary\_green - binary\_blue$$

$$Blue\_only = binary\_blue - binary\_green - binary\_red$$

$$Green\_only = binary\_green - binary\_red - binary\_blue$$

$$Sky\_blue = binary\_green + binary\_blue - binary\_red$$

$$Pink = binary\_blue + binary\_red - binary\_green$$

$$Yellow = binary\_red + binary\_green - binary\_blue$$

Binary images after segmentation will no more be binary when above equations are used for the purpose. These segmented images are thus threshold with thresholding limit depending on the equation used for the purpose.

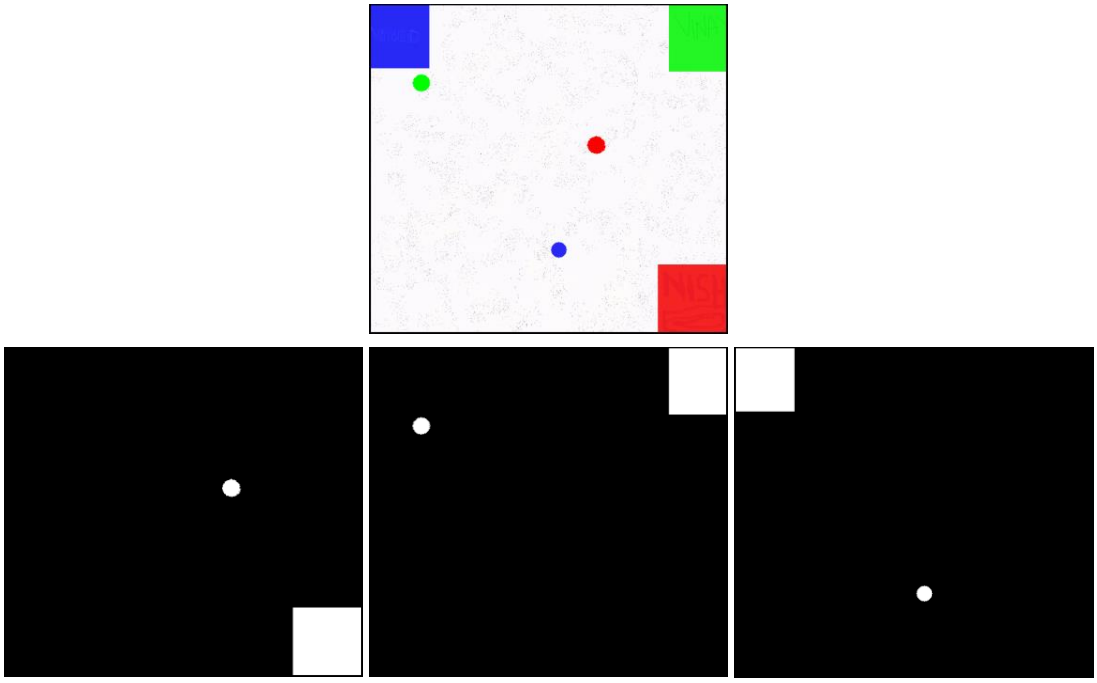


Figure 4.4 Segmented and threshold Red, blue and green component images

4.2.5 MORPHOLOGICAL PROCESSING

After segmentation and thresholding we have images showing us the regions of interest. While plying with the real-time images due to noise and distortion we have few more regions along with region of interest which changes from one acquired frame to another.

Figure 4.5 real time test image and threshold RED color space component

As you can see, the binary image obtained above has quite a lot of noise. It is needed to smoothen the edges, remove the tiny dots scattered here and there so that at last we have some countable number of objects to work upon.

Here we perform morphological closing operation. This operation fills in the gaps between two objects and smoothens the edges. The degree and type of smoothening and joining depend on the structuring element, i.e., the basic shape which is used to perform the operation. The structuring element can be a disk, a diamond, a line, etc.

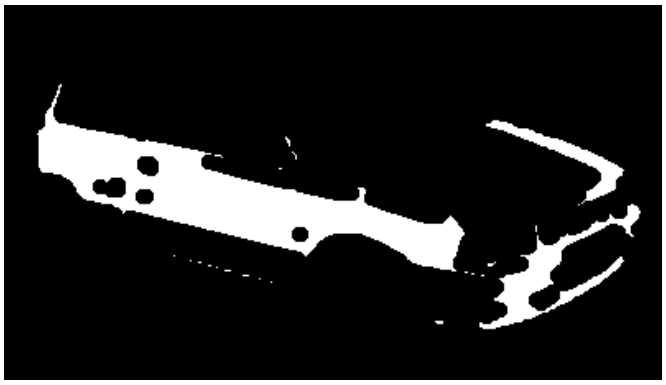


Figure 4.6 morphologically closed image

The next step is to remove holes from the image. Holes are background pixels surrounded by foreground (image) pixels.

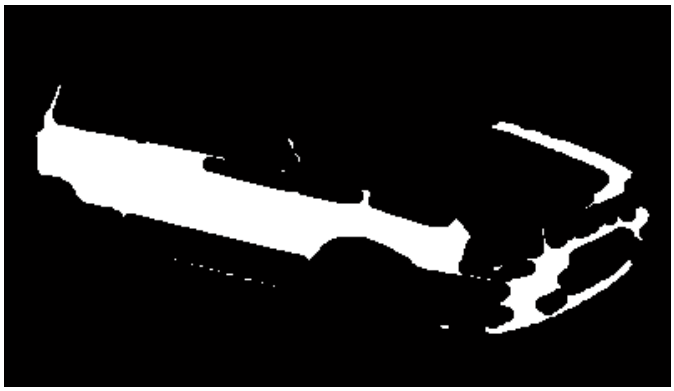


Figure 4.7 image from figure 4.6 with holes removed

Now we need to remove the spatters. From the image depicted by figure 4.7.

Figure 4.8 image from figure 4.7 with splatters removed

The sequence of performing these operations is very significant, as the subsequent operation is performed on the converted image and not the original image. The sequence adopted in the example is not necessary, and it varies in context with the properties of objects under consideration in the images.

The size and shape of structuring element is also to be determined experimentally, so that the number of objects you get in the binary image is same as what we require.

This final image obtained in this example is still not workable, and must be processed further or in a different way to get a consolidated image.

4.3 INFORMATION EXTRACTION

In most of problem statements of robotics based on image processing, we are required to find the centroid, area, and no. of objects of a particular colour. Now we first take a sample image of arena, note down the RGB values and then extract the regions of different colour by making binary images. Then we can find the centre of different objects. The object, destination and the robot can be differentiated on the basis of their area. Orientation of robot can be determined based on orientation of centroids of two tags of different areas on it.

The label matrix is then found corresponding to each component image. A label matrix (L) corresponding to binary image is a 2D matrix of the same size as that of the image. Each object in the binary image is numbered 1, 2, 3… and all the pixels of L corresponding to the objects in binary image have value respectively 1, 2, 3… The background pixels are 0 by default. In other words, the region in L corresponding to first object in the image is marked 1, corresponding to second object is marked 2 and so on.

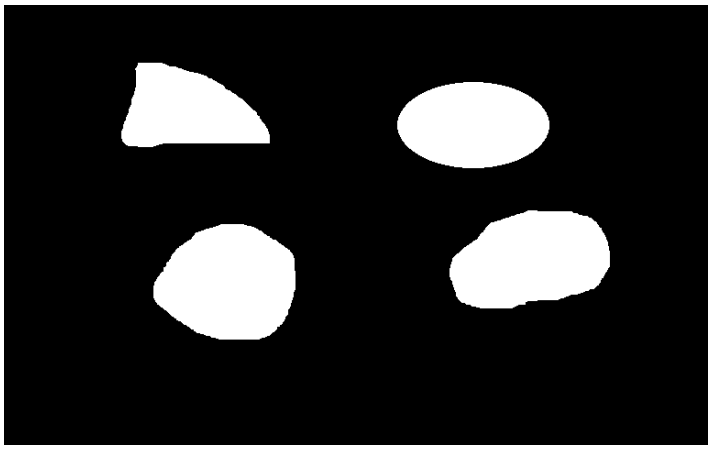


Figure 4.9 first test image for information extraction

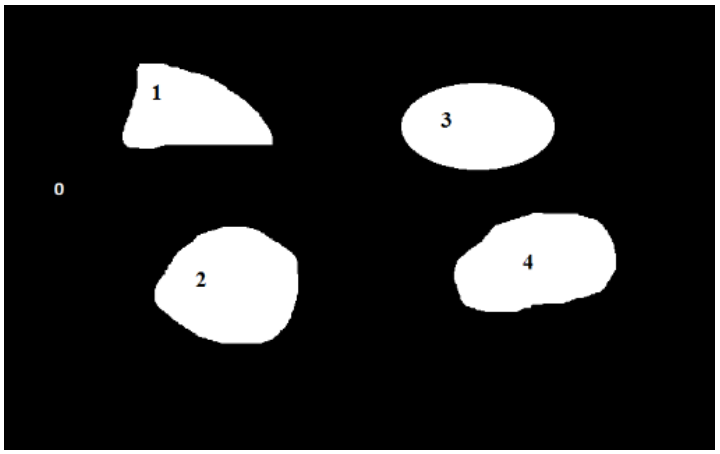


Figure 4.10 label matrix corresponding to figure 4.9

Once we have an image consisting of all the objects labelled, we must now know the centroid, area, etc. of each.

4.4 COMMANDS FOR ROBOT

After finding area of each objects labelled, now we must determine the colour of the object and hence the destination. Red Blue and Sky blue Colour space components whose ever number of objects of significant area are 2, the object and destination are of corresponding color. Hence we operate on that binary image.

Area of object is lesser than that of the destination. Area of head and tail of robot (blue color) also differs. Thus orientations of robot and robot-destination path are found and commands are sent to robot to align both in same direction… and then move forward towards the destination.

In the first stage object will be considered the destination once the object is retrieved, actual destination of object will be destination of robot-object entity.

4.5 CONCLUSION

Extraction of information from pre-processed image and thus data for movement of robot around arena are discussed in this process

CHAPTER 5

COMMUNICATION

5.1 INTRODUCTION

Communication has also a very important role to play in the project, as we need user and central system to communicate and central system and the pod cars have to communicate. We need an efficient communication system.

This communication can be wired or wireless.

- Wired Communication.
- Wireless Communication

5.2 COMMUNICATION SYSTEM

In the previous chapter we have gone through computation on colorspace components of acquired image to obtain presence and position of the object, destination robot and its orientation and thus prepared the commands to be sent to robot to move accordingly. This chapter deals with how we managed to send these commands from Central software (MATLAB) to the robot.

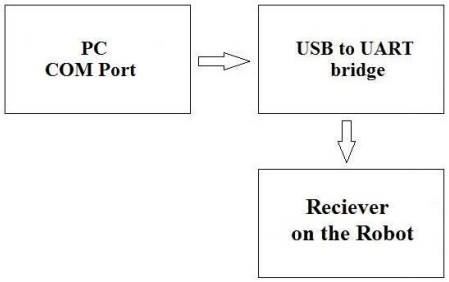


Figure 5.1 Block diagram of Communication system

The commands from Matlab are sent Away from PC to the robot using the system as mentioned in the above block diagram.

5.3 USB to UART Bridge

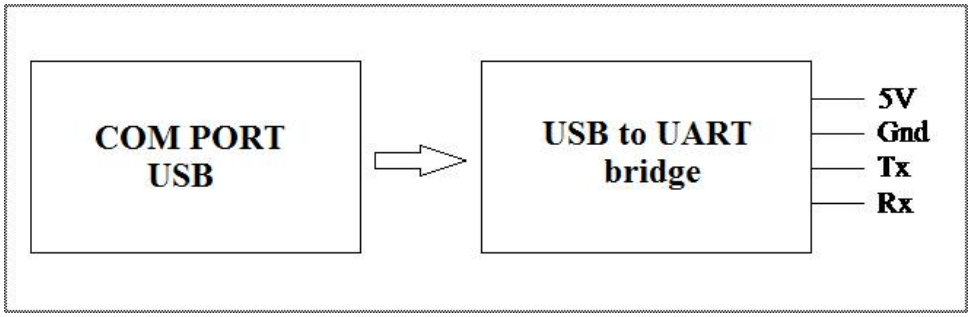


Figure 5.3 Block diagram of USB to UART Bridge

Signal levels of the data being sent through the USB COM port are not compatible with the TTL level microcontroller circuit. Moreover, USB signal is just a set of serial interface protocols which supports 127 different devices to be connected to computer as designed in mid1990's. Hence we use a USB to UART Bridge to interface PC/Laptop with TTL level circuits.

The communication system as mentioned before might be wired or wireless. In case of wired communication, there will be a physical electrical connection between transmitter pin of Serial converter and the receiver pin of the Receiver on Robot. In case if wireless communication, we need a system to encode the data (commands from Matlab) and later modulate the RF carrier to be sent to the receiver where RF signal received is demodulated and decoded to get back the commands.

Instead of an encoder and decoder pair in this prototype project we used a microcontroller in transmitting end to receive the data from UART Bridge and reconnect

to an RF transmitter module of carrier frequency 433MHz whose sister circuit the superheterodyne RF receiver is connected to microcontroller of the Robot.

5.4 USART

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. Here we use USART of the microcontroller on the robot to receive the commands. In case of wired communication we use another microcontroller to communicate with the UART device using USART and then RF module comes into the picture where receiver still uses USART communication to receive data from RF receiver.

5.5 MODES OF OPERATION OF USART

The USART of the AVR can be operated in three modes, namely

- Asynchronous Normal Mode
- Asynchronous Double Speed Mode
- Synchronous Mode

Each of these modes are described below.

ASYNCHRONOUS NORMAL MODE

In this mode of communication, the data are transmitted/received asynchronously, i.e. we do not need (and use) the clock pulses, as well as the XCK pin. The data is transferred at the BAUD rate, we set in the UBBR register. This is similar to the UART operation.

ASYNCHRONOUS DOUBLE SPEED MODE

This is the higher speed mode for asynchronous communication. In this mode also we set the baud rates and other initializations similar to Normal Mode. The difference is that data is transferred at double the baud we set in the UBBR Register. Setting the U2X bit in UCSRA register can double the transfer rate. Setting this bit has effect only for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however, that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.

SYNCHRONOUS MODE

This is the USART operation of AVR. When Synchronous Mode is used (UMSEL = 1 in UCSRC register), the XCK pin will be used as either clock input (Slave) or clock output (Master).

5.6 WIRED COMMUNICATION

Since it is our intention to work with USB to UART Bridge, we will be using Asynchronous Transmission Mode.

Before we transmit the information, we need to set the baud rate, clock frequency, number of data bits and number of parity, start and stop bits to be used same as that of the parameters set in Matlab. Start and/or stop bits are set for every set of data because USART is working on asynchronous mode.

The moment data is sent through the USB COM port, microcontroller stores the information in USART receive buffer. In the case where communication is through wired channel (physical electrical connection), the stored information is retrieved through the program written in microcontroller and robot moves accordingly. Each time transmission is done, the buffer gets overwritten but when no transmission prevails, the previous value is retained. Hence each actuation based data transmission is succeeded by the stop data.

5.7 WIRELESS COMMUNICATION

5.7.1 WIRELESS TRANSMISSION

In case of wireless communication, transmitting pin of USB to UART Bridge is physically connected to USART receiver pin of microcontroller whose transmit pin is connected to RF transmitter which generates RF equivalent signal. This analogue signal is transmitted isotropically all around RF transmitter within the range.

5.7.1 WIRELESS RECEPTION

In case of wireless communication, data pin of receiver is connected to receiver pin of robot’s microcontroller. The demodulated data from the RF receiver is stored in the

USART buffer of the microcontroller on robot and is used for actuation of the robot through the program in flash memory and robot moves accordingly.

5.8 CONCLUSION

Communication has very important role to play in this project, as the robot works only on the basis of commands communicated by the central system. The wired and wireless processes of communication of central system and robot on the arena have been discussed.

CHAPTER 6

HARDWARE DESCRIPTION

6.1 INTRODUCTION

As modern embedded systems grow in complexity component-based development is an increasingly attractive approach utilized to make the development of such systems simpler and less error prone. However, in embedded system domain component-based approach to software development is seldom used in practice, and is mostly explored in component models used in the research context.

Amongst other things, one of the problems that component-based development for embedded systems must address is interaction of a software system with the environment; the physical world that the system is embedded. This interaction is done using hardware devices, such as sensors and many devices depending on the application like camera and actuators. A simple example of an embedded system is a temperature regulation system, which keeps a constant temperature in a room, cooling or heating the air in it. Such a system must have at least one temperature sensor, and two actuators: one for starting the heating process and one for starting the cooling process. So, even if complexity of software in such a system is very low, its behaviour is highly dependent on the communication with hardware devices, and behaviour of the devices themselves.

Communication between software and hardware devices can be as simple as writing a value to a hardware pin or port of the device that the system is deployed on, or as complex as an invocation of a service on a remote device. In all cases, this interaction with the environment implies dependencies of software components on the hardware or

middleware used to communicate with the environment. Same environment and a combination of hardware and middleware also affect the behaviour of an embedded system. As reusability and analysability of software components and component based systems highly rely on such dependencies and effects on behaviour, failure to adequately express they can hinder the use of component-based approach in the embedded system domain.

Now we have hardware components for construction of the robot and the central system. The description of each component used will be given below.

6.2 USB to UART Bridge

This USB to TTL converter combines the functionalities of the USB-232-1 (USB to Single RS232 Adapter) and TTL-232-1 (Port-powered RS232/TTL converter) allows you to convert USB to TTL/CMOS compatible levels and vice versa.

In this project we have used cp2102 converter.The CP2102 is a highly-integrated USB-to-UART Bridge Controller providing a simple solution for updating RS-232 designs to USB using a minimum of components and PCB space. The CP2102 includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with full modem control signals in a compact 5 x 5 mm MLP-28 package. No other external USB components are required.

However because of its small size, and it is also an MLP, a certain degree of technical difficulty welding. So we designed of this compact USB-UART module, leads to the interface including 5V, TXD, RXD, GND, CTS, RTS, which TXD, RXD can directly

connected to the MCU serial port, RXD to MCU-TXD, TXD to MCU-RXD , it can also be connected to the Bluetooth module, GPS and other serial devices, CTS and RTS handshake signals as for special occasions, usually you needless, it easy for the initial product debugging. LED mounted on the drive after the often shiny. The module uses USB male seat, can be connected directly to PC USB seat can also be connected to a USB extension line.

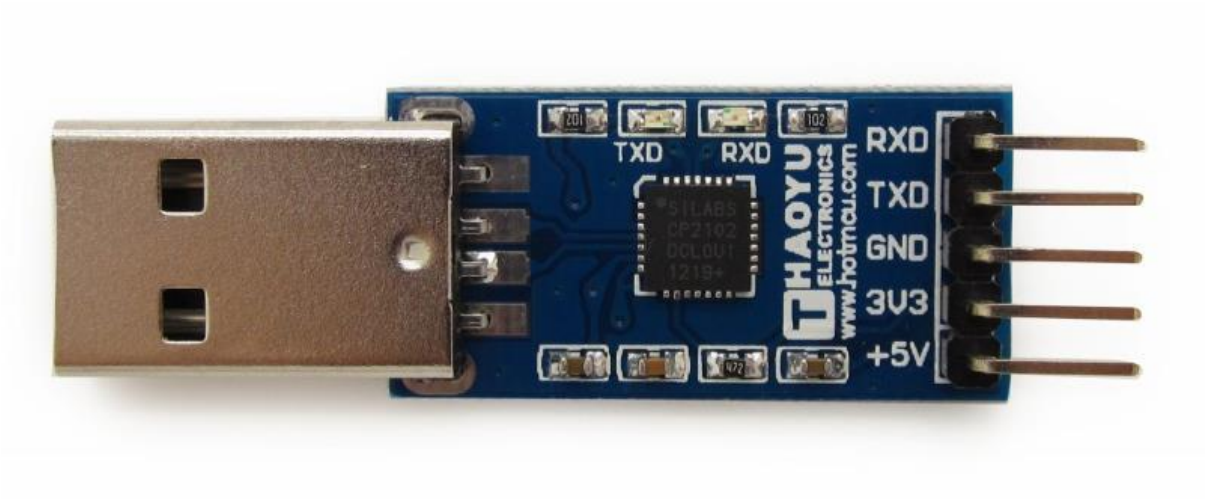


Figure 6.1 USB to UART Bridge

6.3 RF MODULE

The RF module, as the name suggests, operates at Radio Frequency. The corresponding frequency range varies between 30 kHz & 300 GHz. In this RF system, the digital data is represented as variations in the amplitude of carrier wave. This kind of modulation is known as Amplitude Shift Keying (ASK).

Transmission through RF is better than IR (infrared) because of many reasons. Firstly, signals through RF can travel through larger distances making it suitable for long range applications. Also, while IR mostly operates in line-of-sight mode, RF signals can travel

even when there is an obstruction between transmitter & receiver. RF transmission is stronger and reliable than IR transmission. RF communication uses a specific frequency unlike IR signals which are affected by other IR emitting sources.

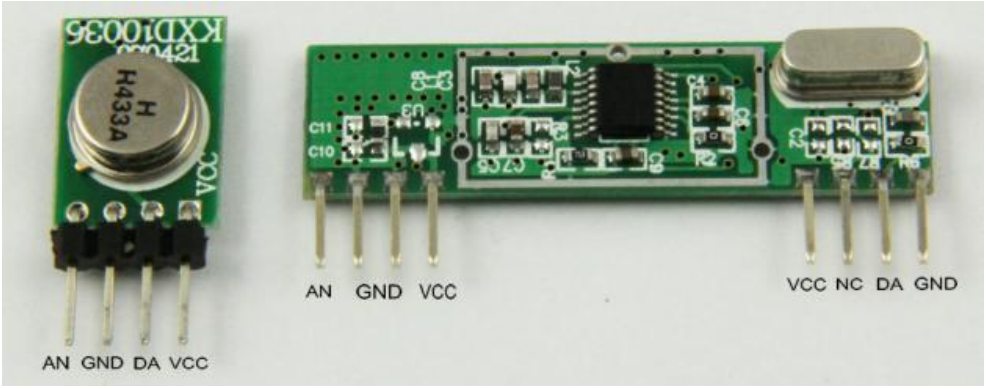


Figure 6.2 433MHz Superheterodyne3400 RF Transmitter and Receiver

This RF module comprises of an RF Transmitter and an RF Receiver. The transmitter/receiver (Tx/Rx) pair operates at a frequency of 315/434 MHz. An RF transmitter receives serial data and transmits it wirelessly as a radio frequency signal through its antenna connected at pin4. The transmission occurs at the rate of 1Kbps - 10Kbps.The transmitted data is received by an RF receiver operating at the same frequency as that of the transmitter.

Transmitter Specifications:

- Working voltage:3V~12V
- Working current:max≤40mA (12V), min≤9mA(3V)
- Modulation mode:ASK/OOK
- Working frequency: 315MHz - 433.92MHz, customized frequency is available.
- Transmission power:25mW (315MHz at 12V)

- Frequency error:+150kHz (max)
- Baud Rate:≤10Kbps
- Aerial Length:24cm (315MHz), 18cm(433.92MHz)

Receiver Specifications:

- Working voltage:5.0VDC +0.5V
- Working current:≤2.5mA (5.0VDC)
- Bandwidth: 2MHz (315MHz, having result from testing at lowing the sensitivity 3dBm)
- Sensitivity: excel –105dBm (50Ω)
- Output signal: TTL electric level signal entire transmit

6.4 MICROCONTROLLER

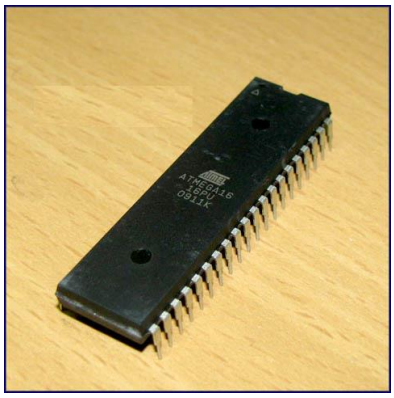


Figure 6.3 Microcontroller – Atmega16

Microcontrollers are "special purpose Computers". Any device that measures, stores, controls, calculates, or displays information is a candidate for putting a microcontroller inside. The microcontroller includes a CPU, RAM, ROM, I/O ports, and timers like a standard computer. Microcontrollers have become common in many areas, and can be

found in home appliances, computer equipment, and instrumentation. They are often used in automobiles, and have many industrial uses as well, and have become a central part of industrial robotics. Because they are usually used to control a single process and execute simple instructions, microcontrollers do not require significant processing power. Microcontrollers are hidden inside a surprising number of products these days. If your microwave oven has an LED or LCD screen and a keypad, it contains a microcontroller

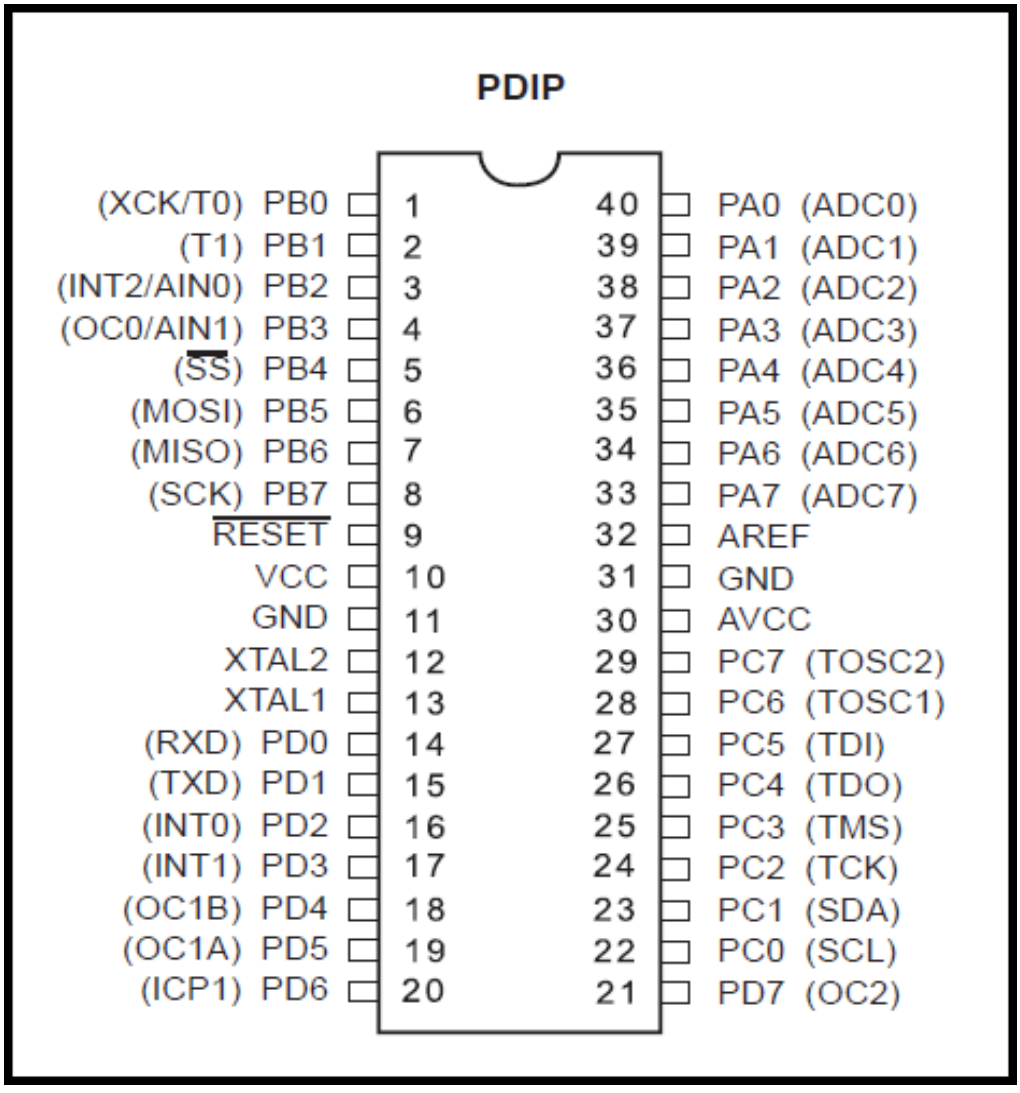


Figure 6.4 Pin description of ATmega16

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
    - 131 Powerful Instructions – Most Single-clock Cycle Execution
    - 32 x 8 General Purpose Working Registers
    - Fully Static Operation
    - Up to 16 MIPS Throughput at 16 MHz
    - On-chip 2-cycle Multiplier
- Non-volatile Program and Data Memories
    - 16K Bytes of In-System Self-Programmable Flash
        Endurance: 10,000 Write/Erase Cycles
    - Optional Boot Code Section with Independent Lock Bits.
        In-System Programming by On-chip Boot Program
        True Read-While-Write Operation
    - 512 Bytes EEPROM
        Endurance: 100,000 Write/Erase Cycles
    - 1K Byte Internal SRAM
    - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
    - Boundary-scan Capabilities According to the JTAG Standard
    - Extensive On-chip Debug Support
    - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface

- Peripheral Features
    - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
    - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
    - Real Time Counter with Separate Oscillator
    - Four PWM Channels
    - 8-channel, 10-bit ADC
        - 8 Single-ended Channels
        - 7 Differential Channels in TQFP Package Only
        - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
    - Byte-oriented Two-wire Serial Interface
    - Programmable Serial USART
    - Master/Slave SPI Serial Interface
    - Programmable Watchdog Timer with Separate On-chip Oscillator
    - On-chip Analog Comparator
- Special Microcontroller Features
    - Power-on Reset and Programmable Brown-out Detection
    - Internal Calibrated RC Oscillator
    - External and Internal Interrupt Sources
    - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
    - 32 Programmable I/O Lines
    - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF

- Operating Voltages
    - 2.7 - 5.5V for ATmega16L
    - 4.5 - 5.5V for ATmega16
- Speed Grades
    - 0 - 8 MHz for ATmega16L
    - 0 - 16 MHz for ATmega16
- Power Consumption @ 1 MHz, 3V, and 25°C for ATmega16L
    - Active: 1.1 mA
    - Idle Mode: 0.35 mA
    - Power-down Mode: < 1 μA

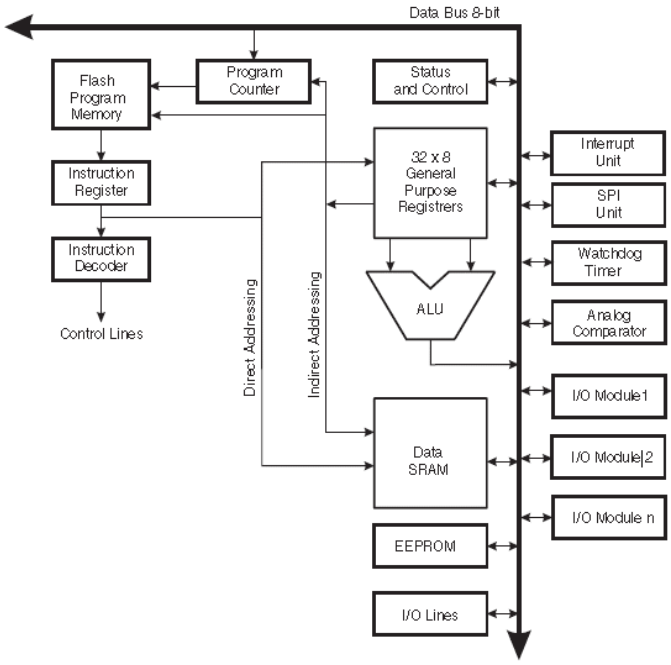


Figure 6.5 AVR CPU architecture

CISC

Pronounced sisk, and stands for Complex Instruction Set Computer. Most PC's use CPU based on this architecture. For instance Intel and AMD CPU's are based on CISC architectures. Typically CISC chips have a large amount of different and complex instructions. The philosophy behind it is that hardware is always faster than software, therefore one should make a powerful instruction set, which provides programmers with assembly instructions to do a lot with short programs.

RISC

Pronounced risk, and stands for Reduced Instruction Set Computer. RISC chips evolved around the mid-1980 as a reaction at CISC chips. The philosophy behind it is that almost no one uses complex assembly language instructions as used by CISC, and people mostly use compilers which never use complex instructions. Apple for instance uses RISC chips. Therefore fewer, simpler and faster instructions would be better, than the large, complex and slower CISC instructions. However, more instructions are needed to accomplish a task.

Another advantage of RISC is that - in theory - because of the more simple instructions, RISC chips require fewer transistors, which makes them easier to design and cheaper to produce. Finally, it's easier to write powerful optimized compilers, since fewer instructions exist.

RISC Vs CISC

There is still considerable controversy among experts about which architecture is better. Some say that RISC is cheaper and faster and therefore the architecture of the

future. Others note that by making the hardware simpler, RISC puts a greater burden on the software. Software needs to become more complex. Software developers need to write more lines for the same tasks.

Therefore they argue that RISC is not the architecture of the future, since conventional CISC chips are becoming faster and cheaper anyway. RISC has now existed more than 10 years and hasn't been able to kick CISC out of the market. If we forget about the embedded market and mainly look at the market for PC's, workstations and servers I guess a least 75% of the processors are based on the CISC architecture. Most of them the x86 standard (Intel, AMD, etc.), but even in the mainframe territory CISC is dominant via the IBM/390 chip.

Looks like CISC is here to stay … Is RISC than really not better? The answer isn't quite that simple. RISC and CISC architectures are becoming more and more alike. Many of today's RISC chips support just as many instructions as yesterday's CISC chips. The PowerPC 601, for example, supports more instructions than the Pentium. Yet the 601 is considered a RISC chip, while the Pentium is definitely CISC

6.4.1 POWER SUPPLY CIRCUIT

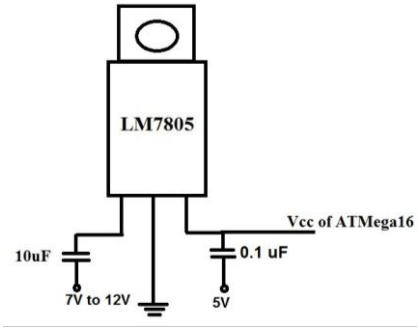


Figure 6.6 Regulated +5V Power supply Circuit

LM 7805 is the heart of regulated power supply circuit.

The LM78XX series of three terminal regulators is available with several fixed output voltages making them useful in a wide range of applications. One of these is local on card regulation, eliminating the distribution problems associated with single point regulation. The voltages available allow these regulators to be used in logic systems, instrumentation, Hi.Fi, and other solid state electronic equipment. Although they are designed primarily as fixed voltage regulators, these devices can be used with external components to obtain adjustable voltages and currents. The LM78XX series is available in an aluminium TO-3 package which will allow over 1.0A load current if adequate heat sinking is provided. Current limiting is included to limit the peak output current to a safe value. Safe area protection for the output transistor is provided to limit internal power dissipation. If internal power dissipation becomes too high for the heat sinking provided, the thermal shutdown circuit takes over preventing the IC from overheating. Considerable effort was expanded to make the LM78XX series of regulators easy to use and minimize the number of external components. It is not necessary to bypass the output, although this does improve transient response. Input bypassing is needed only if the regulator is located far from the filter capacitor of the power supply. For output voltage other than 5V, 12V and 15V the LM117 series provides an output voltage range from 1.2V to 57V. Following are the features of LM7805 regulators

- Output current in excess of 1A
- Internal thermal overload protection
- No external components required
- Output transistor safe area protection

- Internal short circuit current limit
- Available in the aluminium TO-3 package

6.4.2 RESET CIRCUIT

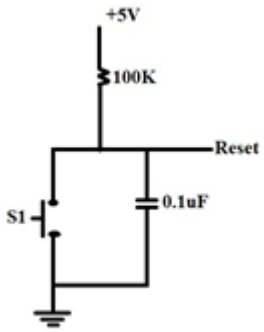


Figure 6.7 Microcontroller reset circuit

A real reset circuit is not necessary in order for a Microcontroller to function in a circuit. In order to set up internal register properly to their initial state after power is applied, there is reset on power-on needed. The only component required to run a Microcontroller, other than those parts that make up the oscillator circuit, is a pull-up resistor connected to the MCLR/ V_{pp} pin. If we omit the pull-up resistor, the Microcontroller will remain in reset (clear) mode on power-up, and will not execute its program. Resistor and pushbutton switch S1 make up an actual reset circuit. When S1 is pressed, it completes a low impedance connection from the MCLR/Vpp pin to ground, forcing the Microcontroller into reset (clear) mode.

6.4.3 CRYSTAL OSCILLATOR CIRCUIT

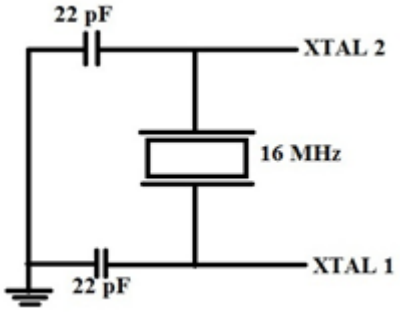


Figure 6.8 Crystal Oscillator Circuit

A crystal oscillator is an electronic circuit that uses the mechanical resonance of a vibrating crystal of piezo electric material to create an electric signal with a very precise frequency. The frequency is commonly used to keep track of time, to provide a stable clock signal for digital integrated circuits. The most common type of piezo electric resonator used is the quartz crystal, so the oscillator circuits designed around them are called Crystal Oscillators.

6.5 DC MOTORS

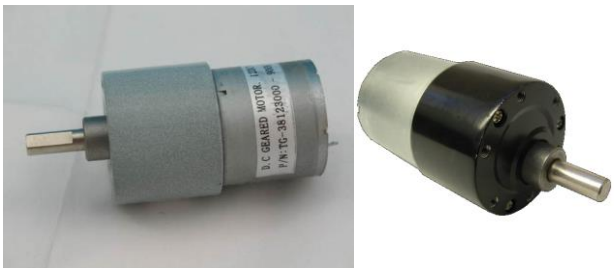


Fig 6.9 DC motors

As a beginner we mostly use DC motors, stepper motor and servo motor will come later. As everybody knows DC motor has two leads. If we apply +ve to one lead and

ground to another motor will rotate in one direction, if we reverse the connection the motor will rotate in opposite direction. If we keep both leads open or both leads ground it will not rotate(but some inertia will be there). If we apply +ve voltage to both leads then braking will occurs. You can test this, first without applying any voltage you rotate the shaft of the motor, then apply ground on both lead and try to rotate the shaft. Both will almost remain same, but if we apply both lead +ve voltage(+12V) and try to rotate the shaft, you can feel the difference between the previous one. You have to apply more force to rotate the same rotation in previous connection. So we take this condition as braking, because if we want to stop the motor suddenly then this is the better way which is easily possible. There are methods to brake motor fastly, like shorting two leads, applying negative polarity exists, but we won't use this in robotics. We apply (1,1) condition to break the motor faster (see H-bridge section for more about it). The main things about a DC motor are Voltage rating, current rating, Torque, Speed. Remember Torque is inversely proportional to speed. So we had to get a good speed motor to get good torque because we can operate the good speed motor in slow speed to get good torque. So maximum speed of the motor should be as high as possible.

6.6 MOTOR DRIVER: L298N

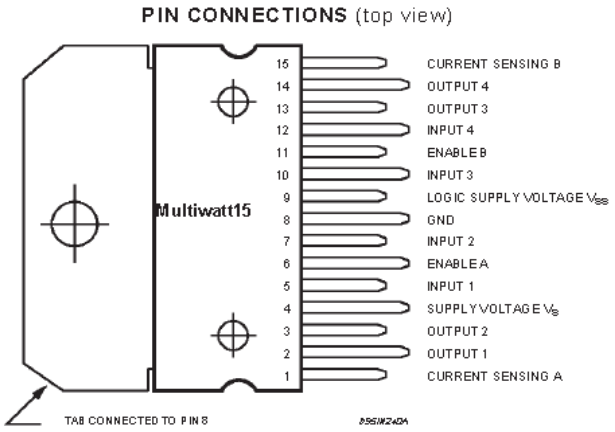


Figure 6.10 Multi-watt Motor driver L298N

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multi-watt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

6.7 CONCLUSION

Complete description of hardware used for the project has been discussed and complete description of ATmega 16 micro controller is also studied.

CHAPTER 7

SOFTWARES USED

7.1 INTRODUCTION

We have used few softwares for programming and transferring the code into microcontroller. This chapter gives idea about how to use the softwares we have used. Softwares used by us are

- Matlab
  - Image Acquisition Toolbox
  - Image Processing Toolbox
- AVR studio for coding
- WIN AVR
- AVR loader, for transferring program into ATmega16
- Proteus 8 for PCB designing and simulation

7.1 ADVANTAGES OF EMBEDDED-C OVER ASSEMBLY

Programs written in assembly can execute faster, while programs written in C are easier to develop and maintain. In traditional applications, such as programs run on personal computers and mainframes, C is almost always the first choice. If assembly is used at all, it is restricted to short subroutines that must run with the utmost speed. For every traditional programmer that works in assembly, there are approximately ten that use C.

A key advantage of using a high-level language (such as C, Fortran, or Basic) is that the programmer does not need to understand the architecture of the microprocessor being used; knowledge of the architecture is left to the compiler. For instance, a short C program uses several variables: n, s, result, plus the arrays: x[ ] and y[ ]. All of these variables must be assigned a "home" in hardware to keep track of their value. Depending on the microprocessor, these storage locations can be the general purpose data registers, locations in the main memory, or special registers dedicated to particular functions. However, the person writing a high-level program knows little or nothing about this memory management; this task has been delegated to the software engineer who wrote the compiler. The problem is, these two people have never met; they only communicate through a set of predefined rules. High-level languages are easier than assembly because you give half the work to someone else. However, they are less efficient because you aren't quite sure how the delegated work is being carried out.

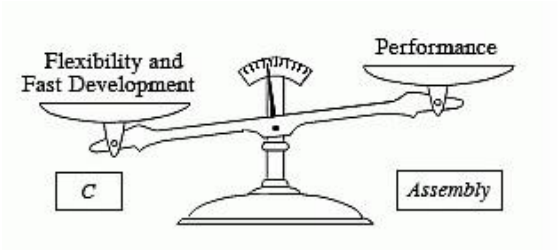


Fig 7.1 Trade-off between Embedded-C and Assembly

7.3 WinAVR

WinAVR is a suite of executable, open source software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform. It includes

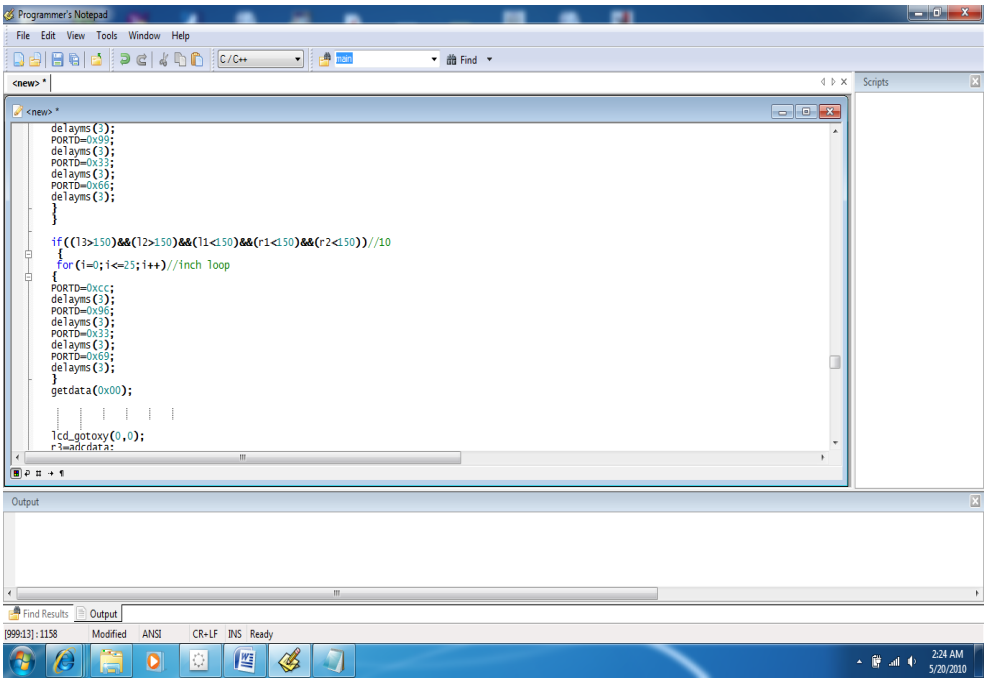
the GNU GCC compiler for C and C++. WinAVR is a collection of executable software development tools for the Atmel AVR processor hosted on Windows.

These software development tools include:

- Compilers
- Assembler
- Linker
- Librarian
- File converter
- Other file utilities
- C Library
- Programmer software
- Debugger
- In-Circuit Emulator software
- Editor / IDE
- Many support utilities

7.4 PROGRAMMER'S NOTEPAD

WinAVR comes with an editor / IDE called Programmers Notepad. This is an Open Source editor with some IDE capabilities. Because the compiler and associated utilities are all command-line driven, we are free to use whatever editor / IDE you want to provide it can call command-line programs.



7.2 Programmer's Notepad

7.5 IN-SYSTEM PROGRAMMING

In-system programming (abbreviated ISP) is the ability of some PLDs, micro controllers, and other programmable electronic chips to be programmed while installed in a complete system, rather than requiring the chip to be programmed prior to installing it into the system.

The primary advantage of this feature is that it allows manufacturers of electronic devices to integrate programming and testing into a single production phase, rather than requiring a separate programming stage prior to assembling the system. This may allow manufacturers to program the chips in their own system's production line instead of buying pre-programmed chips from a manufacturer or distributor, making it feasible to apply code or design changes in the middle of a production run.

Typically, chips supporting ISP have internal circuitry to generate any necessary programming voltage from the system's normal supply voltage, and communicate with the programmer via a serial protocol. Other devices usually use proprietary protocols or protocols defined by older standards. In systems complex enough to require moderately large glue logic, designers may implement a JTAG-controlled programming subsystem for non-JTAG devices such as flash memory and microcontrollers, allowing the entire programming and test procedure to be accomplished under the control of a single protocol.

7.6 AVRstudio 4

AVR Studio is the platform used to run debug the code written for AVR MCU's, it supports all the AVR microcontrollers and the software can be used in the following way.

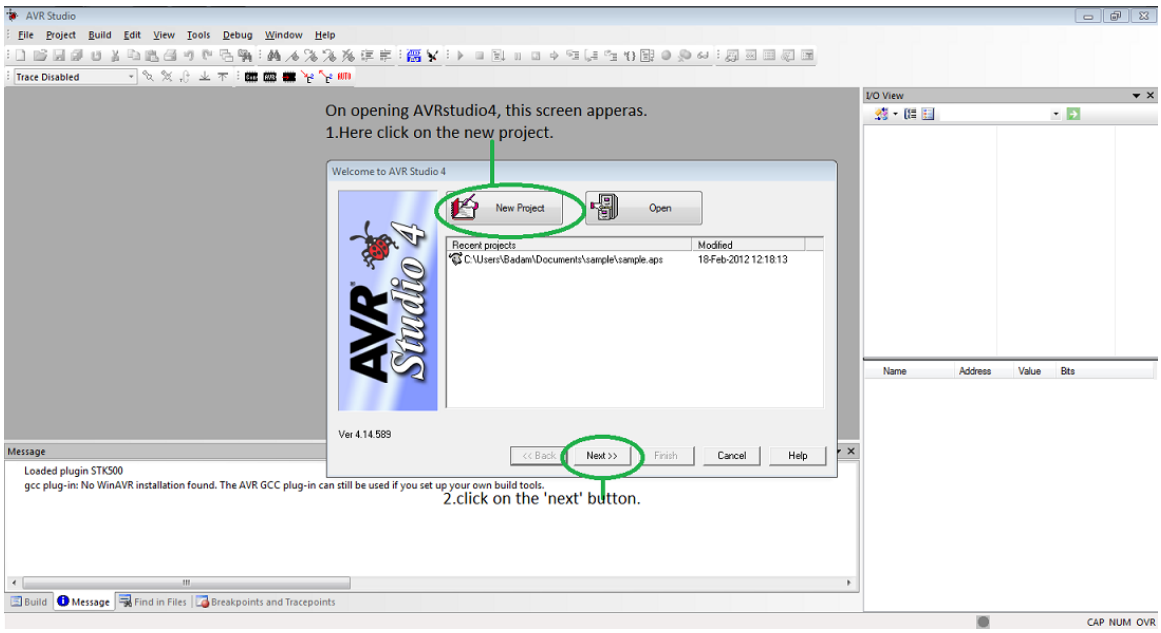
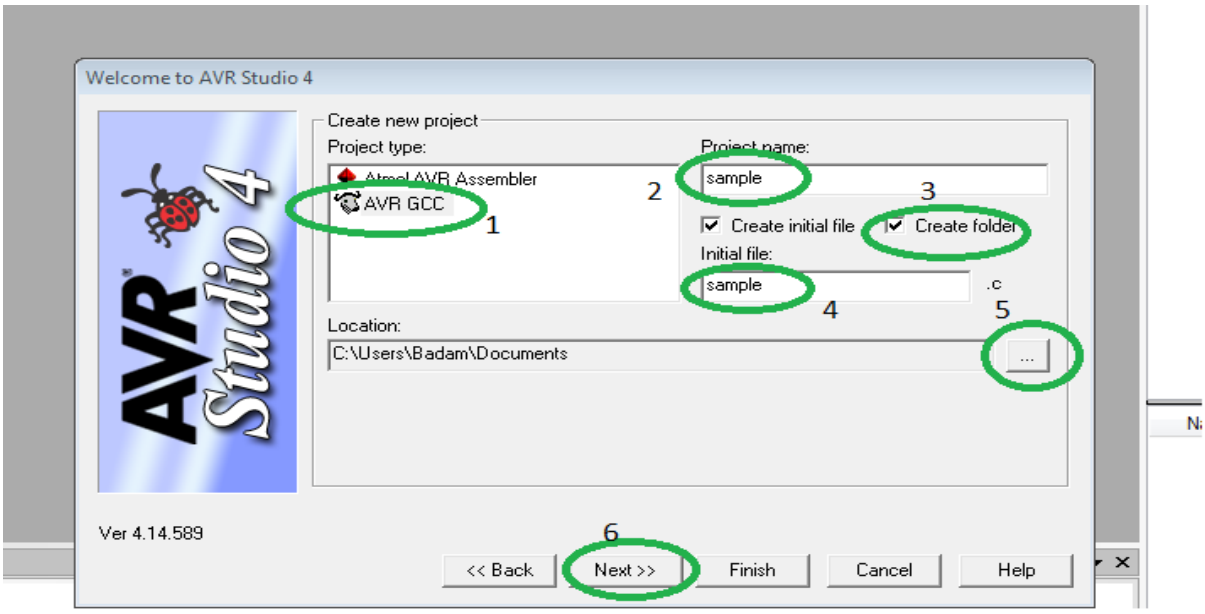
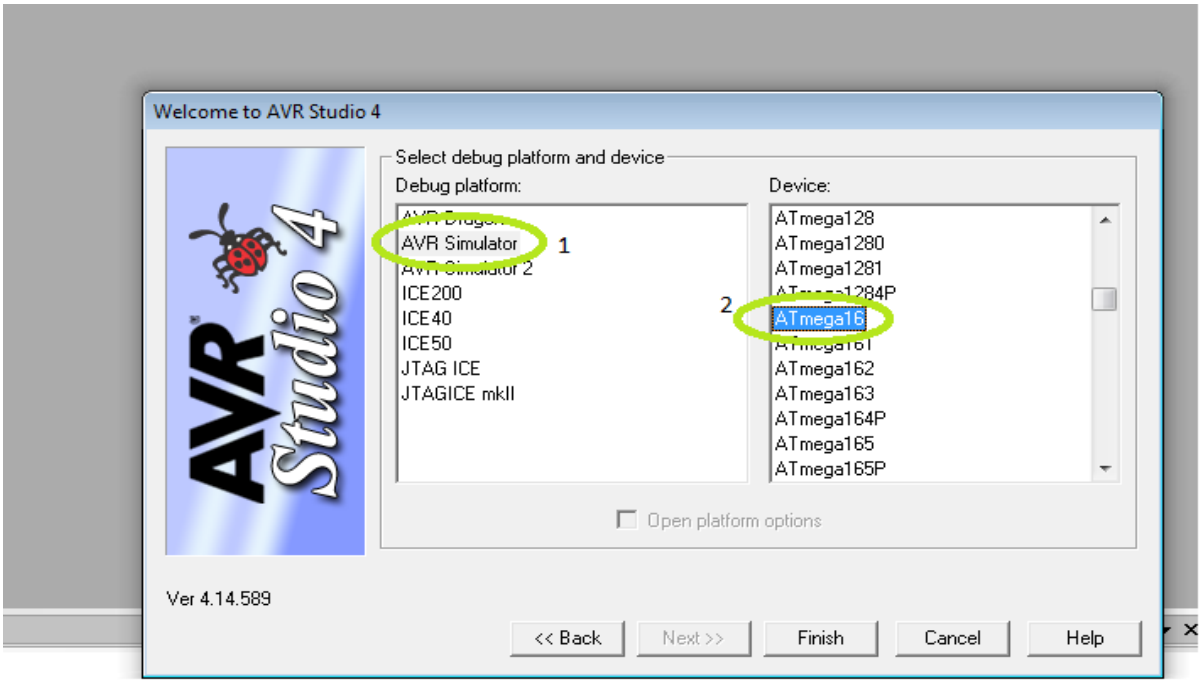


Figure 7.3 AVRstudio 4 Creating a new project.



1. click on AVR GCC, 2. write project name, 3. click on create dolder, 4.it will be generated automatically
5. choose the location where you wanna save file 6. click on next

Figure 7.4 selection of compiler



1. select AVR simulator, 2. scroll down and click on ATmega16

Figure 7.5 selection of simulator and device

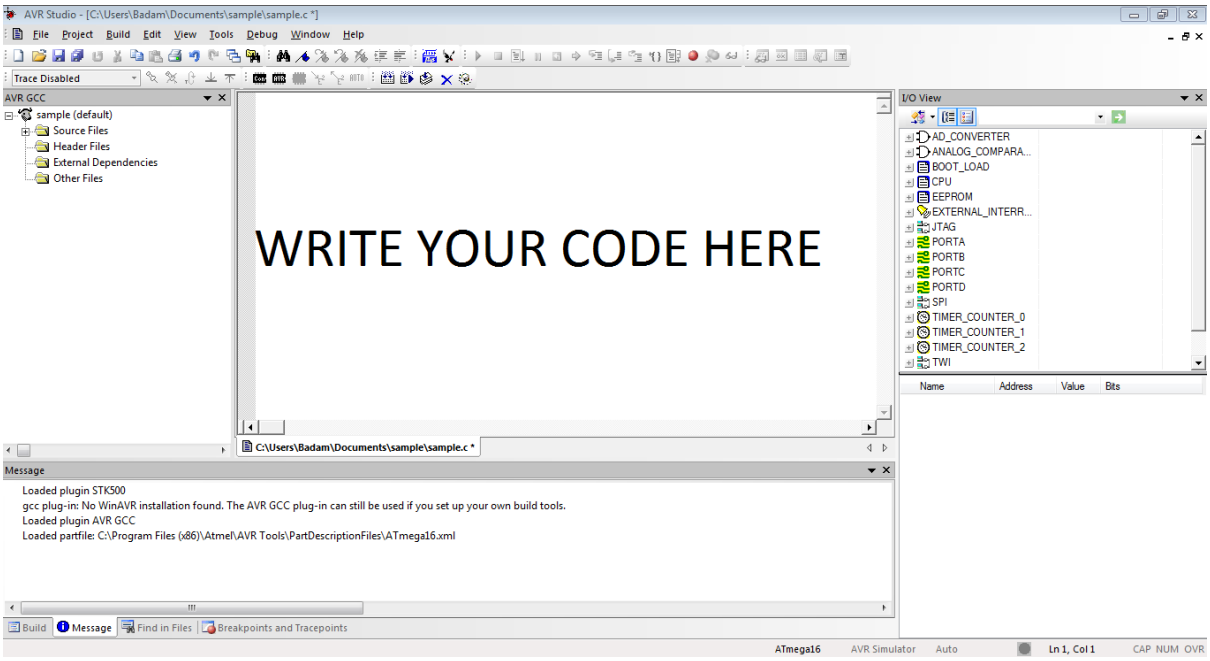


Figure 7.6 Console window to write the code for selected device

To build the code and generate hex file, click on build of build menu on top . The hex code is found in 'default' folder of the project folder.

7.7 AVR Loader

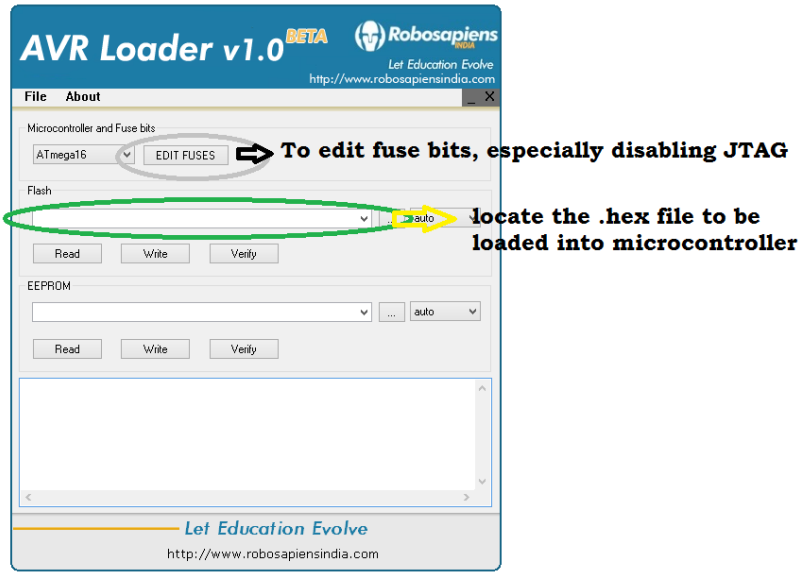


Figure 7.7 AVR Loader for transferring program hex file into microcontroller

7.8 PROTEUS 8

Proteus 8 is a single application with many service modules offering different functionality (schematic capture, PCB layout, etc.). The wrapper that enables all of the various tools to communicate with each other consists of three main parts.

The common database contains information about parts used in the project. A part can contain both a schematic component and a PCB footprint as well both user and system properties. Shared access to this database by all application modules makes possible a huge number of new features, many of which will evolve over the course of the Version 8 lifecycle.

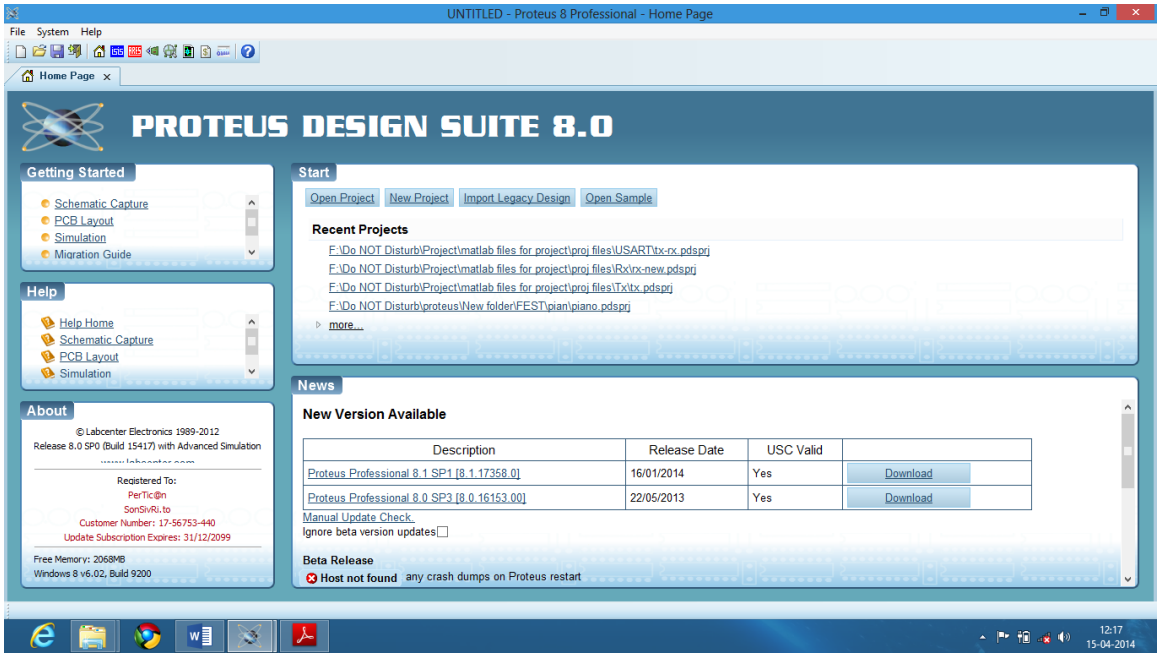


Figure 7.8 (a) PCB design and Circuit simulation software

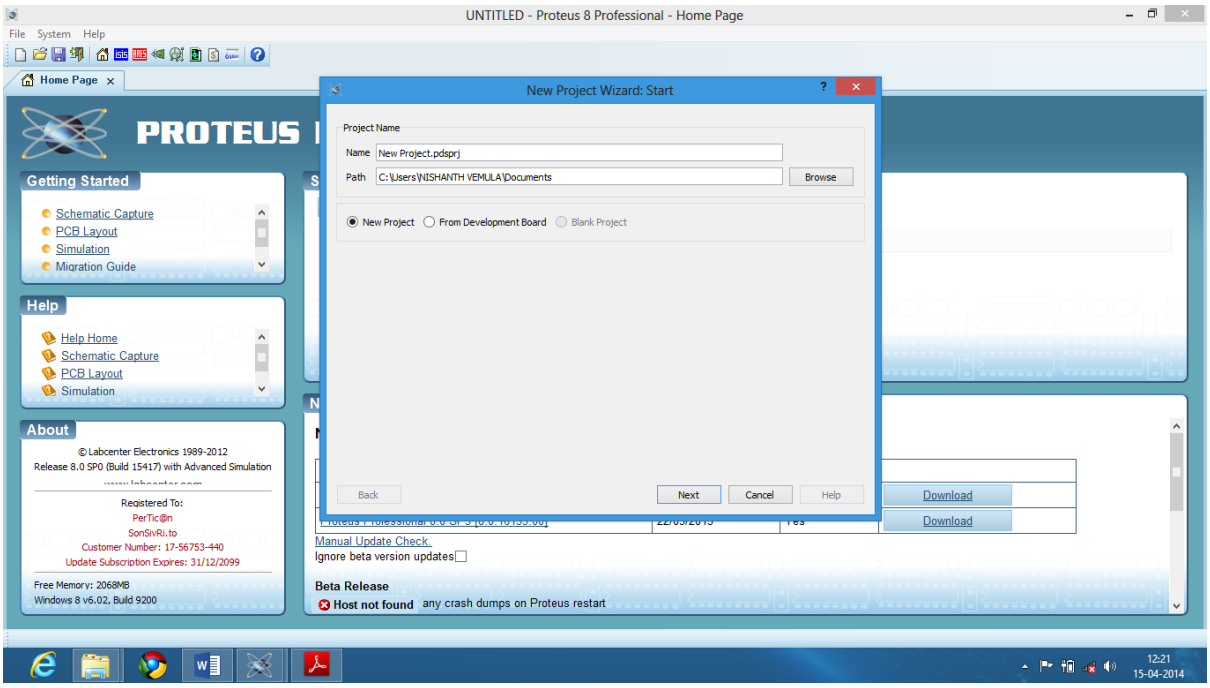


Figure 7.8 (b) PCB design and Circuit simulation software creating new project

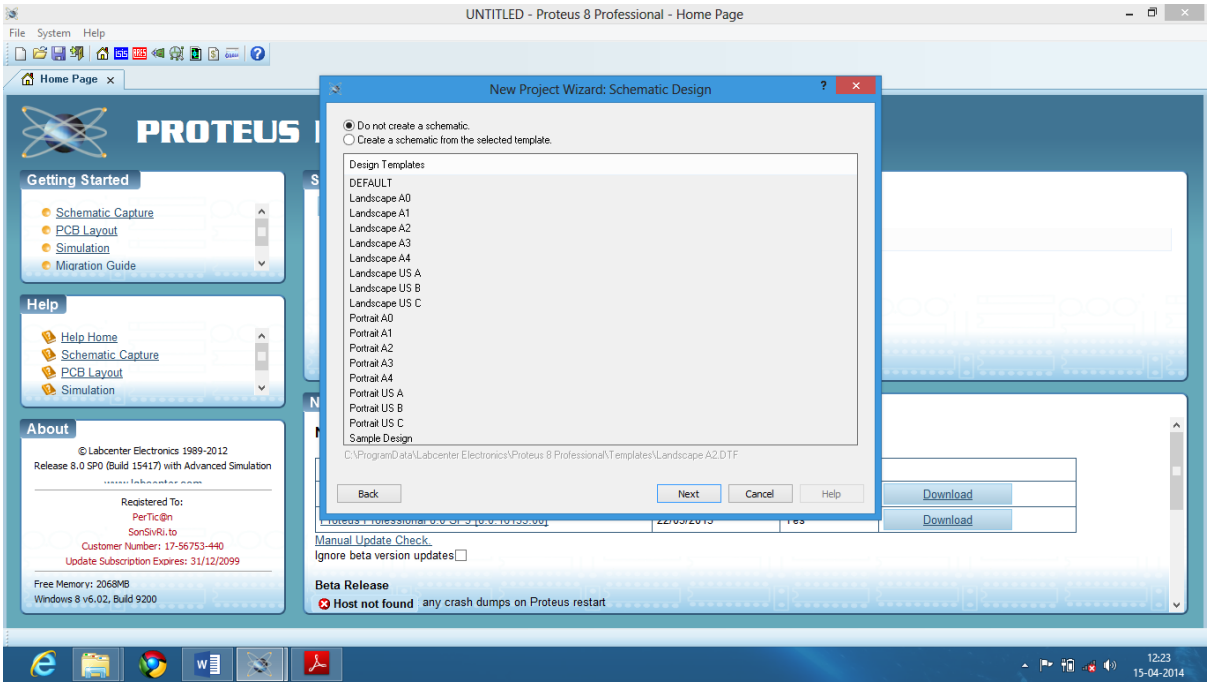


Figure 7.8 (c) PCB design and Circuit simulation software schematic layout panel creation

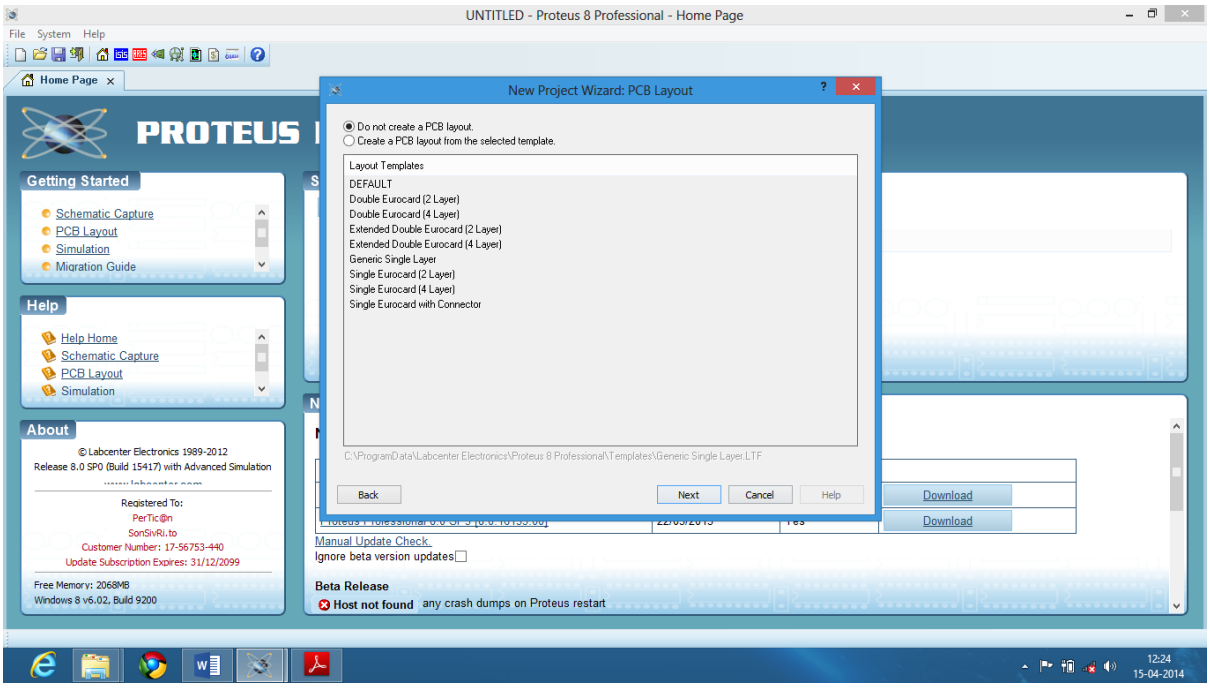


Figure 7.8 (d) PCB design and Circuit simulation software creating PCB layout file

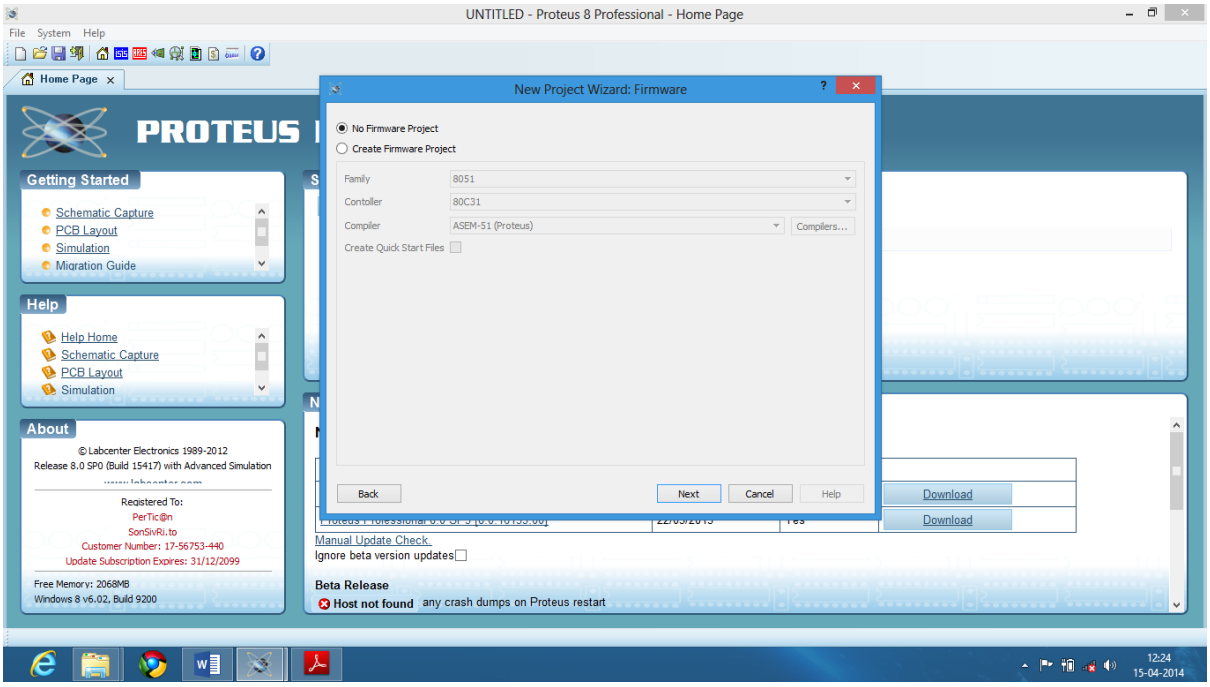


Figure 7.8 (e) PCB design and Circuit simulation software

There is a provision to even write programs for microcontrollers like AVR Studio.

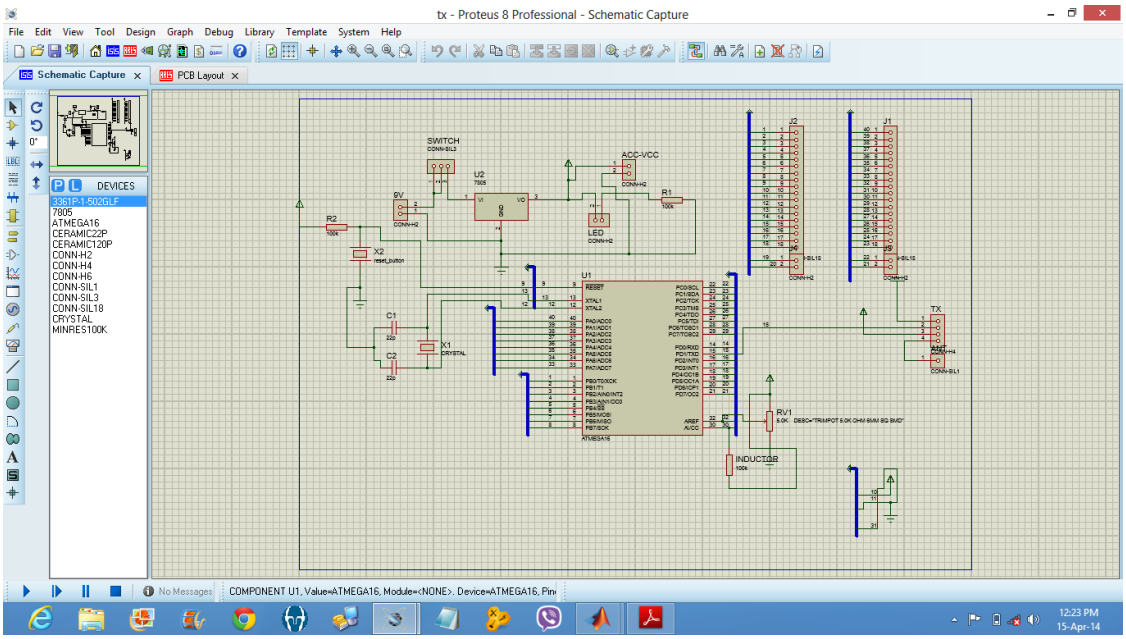


Figure 7.8 (f) Circuit simulation

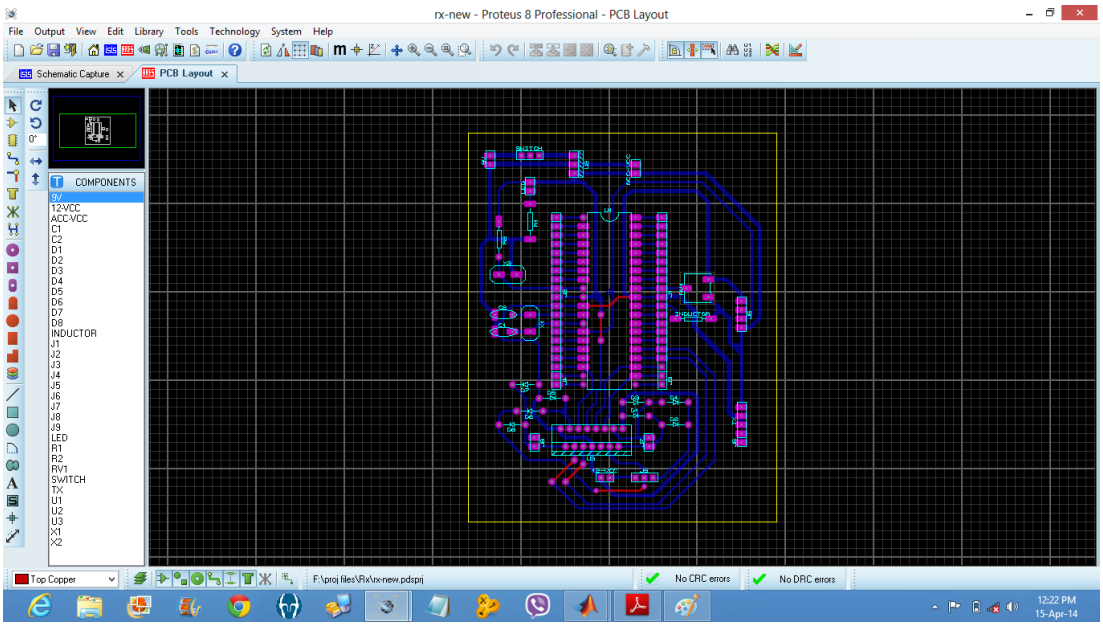


Figure 7.8 (g) PCB design

CHAPTER 8

RESULTS AND CONCLUSION

8.1 INTRODUCTION

The embedded system of the project has been intended to be tested under quasi-stable and uniform lightening conditions which is a best suitable environment for replicating closed room motherboard assembly line.

8.2 RESULTS

After implementation of some or all of the processing techniques on the embedded system based on the illumination conditions at the site of its installation we, the project team, did successfully manage to make the robot move the target object toward its destination.

When it comes to relocation of multiple objects placed on the arena at a single instant of time, we have provided the robot with priorities of which object to be relocated first and which one later. When it is required to relocate multiple objects we could manage to identify certain exceptions resolving of which is held back under scope of future work.

8.1 CONCLUSION

We did successfully manage to design an autonomous embedded system to drive the robot to move single target object flawlessly toward its destination. The system has

been successfully tested for multiple objects under an exception that they are placed significantly far because of lack of pick-place mechanism in the design of the robot.

CHAPTER 9

SCOPE FOR FUTURE WORK

1. If we can do lot of research on this system and overcoming the real time problems, we can bring this system in to reality very soon. The real time problems include the following aspects.
    a. Objects can be identified based on their unique properties. In the prototype we have separated objects by the colour tags. Matlab supports identification of objects based on template matching. By just feeding few templates of each object in different orientations, central system can identify and distinguish each object. For this purpose Computer Vision Toolbox is needed.
    b. A precise pick and place mechanism for allowing relocation of multiple objects has to be incorporated to the robot.
    c. When it comes to almost identical objects with very minute differences, we can use feature extraction techniques of Computer Vision Toolbox.
2. In future when load on the central system increases we need to switch over to a microprocessors instead of microcontrollers
3. In our vision, the future will witness remarkable progress in intelligent assembly system on two key factors.
    a. First, in the near future, the key ideas underlying the intelligent assembly system will extend beyond all limitations and robots to other mechanisms including prosthetic hands.

BIBILOGRAPHY

Davies, E. R. [2005]. *Machine Vision: Theory, Algorithms, Practicalities*, Morgan Kaufmann, San Francisco, CA.

Bezdek, J. C. et al. [2005]. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, Springer, New York.

Umbaugh, S. E. [2005]. Computer Imaging: Digital Image Analysis and Processing, CRC Press, Boca Raton, FL.

Gonzalez, R. C.,Woods, R. E., and Eddins, S. L. [2004]. Digital Image Processing Using MATLAB, Prentice Hall, Upper Saddle River, NJ.

WEBSITES

- www.atmel.com
- www.extremeelectronics.co.in
- www.engineersgarage.com
- www.nex-robotics.com
- www.avrfreaks.com
- www.instrucyables.com

APPENDIX

10.1 ATmega16 BASED RF TRANSMITTER CIRCUIT

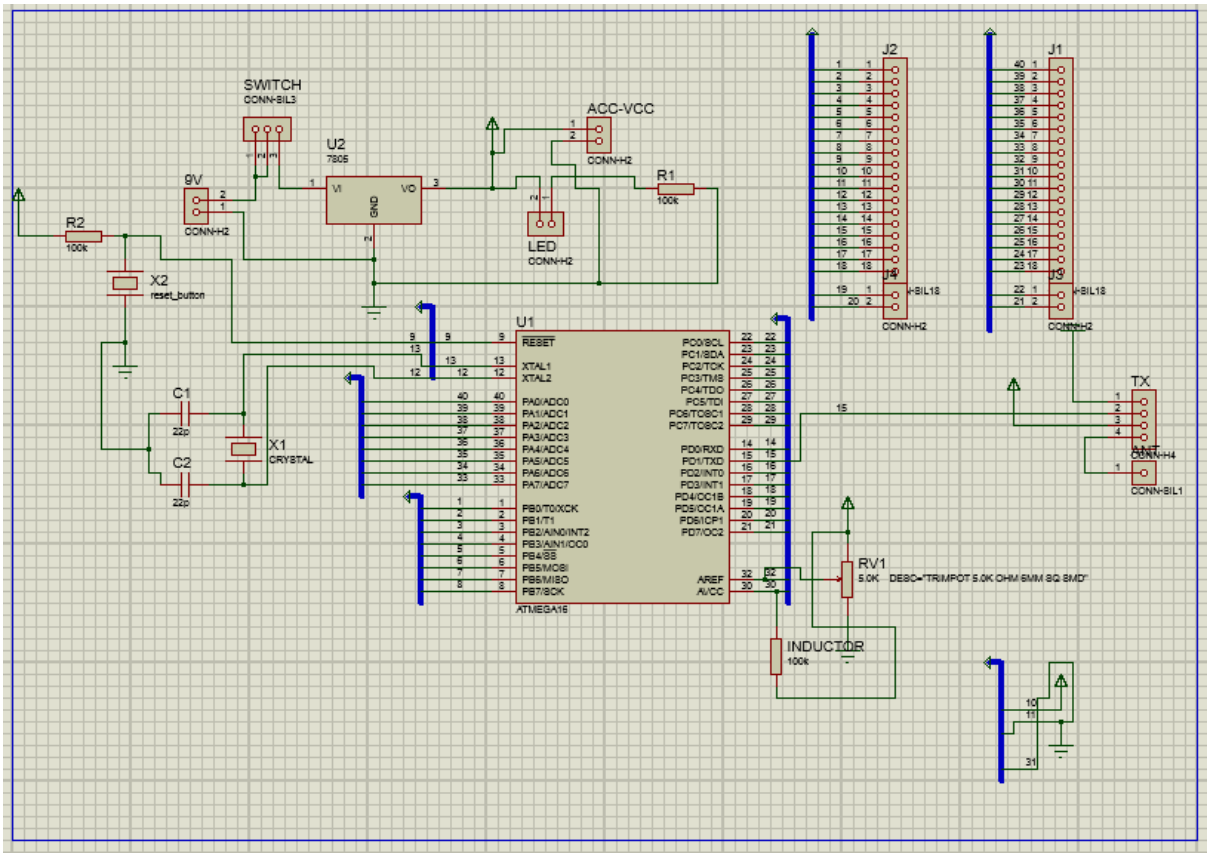


figure 10.1 Transmitter circuit

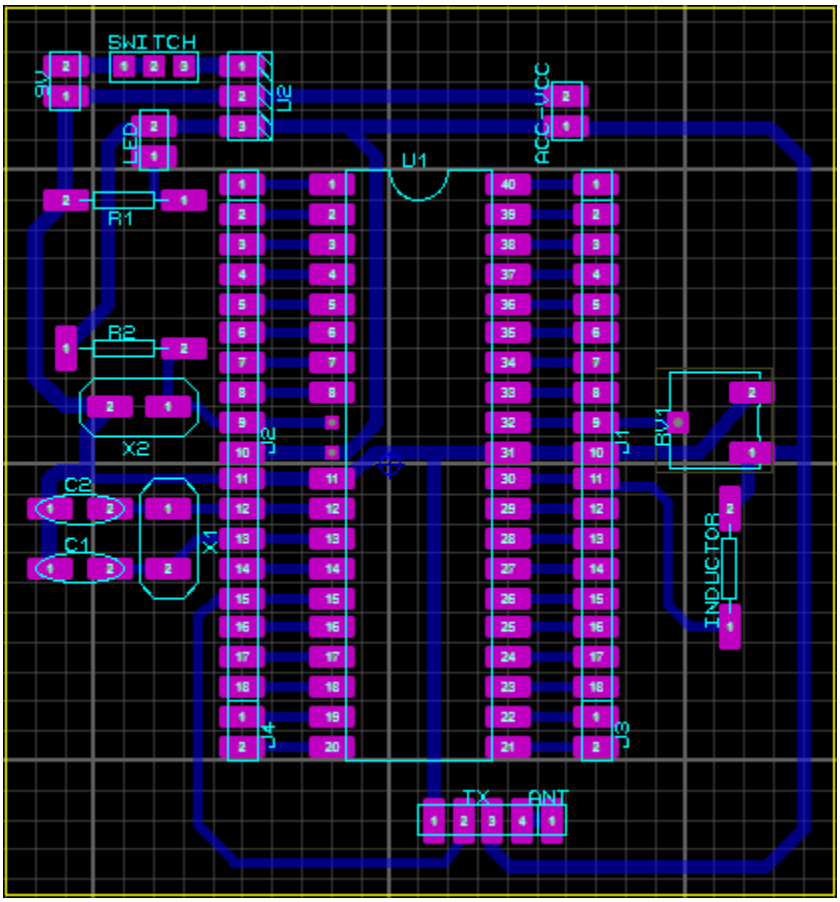


Figure 10.2 Transmitter layout

10.2 ATmega16 BASED RF RECEIVER CIRCUIT

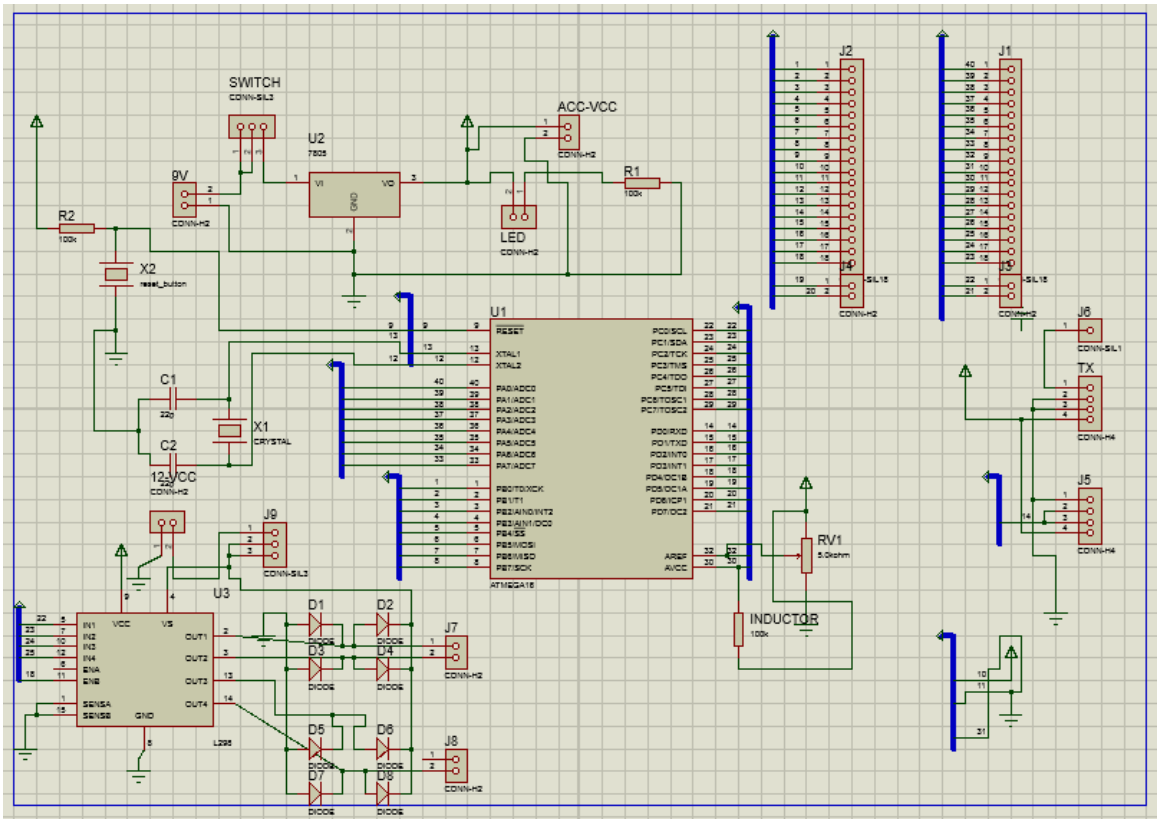


Figure 10.3 Receiver circuit

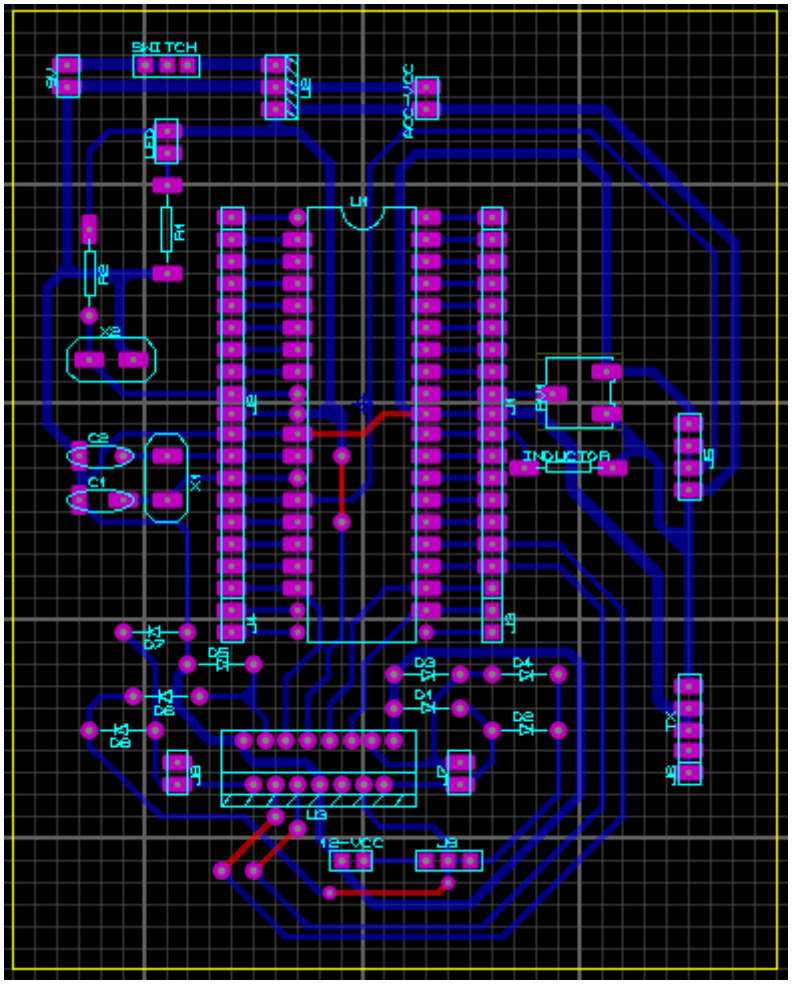


Figure 10.4 Receiver layout

10.3 USB Programmer

This device is specially designed to work with Laptops/Notebooks which doesn't have Parallel or serial port. At full clock speed of 16 MHz of the microcontroller it can program the flash at very high speeds in STK500 mode. This programmer is supported in STK500 as well as Human Interface Device (HID) mode. It is supported on all versions of Windows, including Windows Vista and Longhorn, as well as on Linux.

Features:

- Compatible to Atmel's STK500V2.
- Compatible with AVR Studio, AVRDUDE and compilers having support for STK500V2 Protocol.
- Supports 2 modes, STK500 and USB-HID for compatibility.
- Adjustable ISP clock allows flashing of devices clocked at very low rate, e.g. 32 kHz.
- High Speed Programming : Programs 32 KB flash in just 15 seconds at full speed of Microcontroller.
- ISP clock can be lowered with a jumper (if the programmer software does not support setting the ISP clock)
- Second USB to Serial converter for processing debug output from the target.
- Uses USB power supply, no external supply required.

Supported on Windows 98, XP, Vista and Linux.

10.4 SNAPSHOTS

figure 10.5 Arena

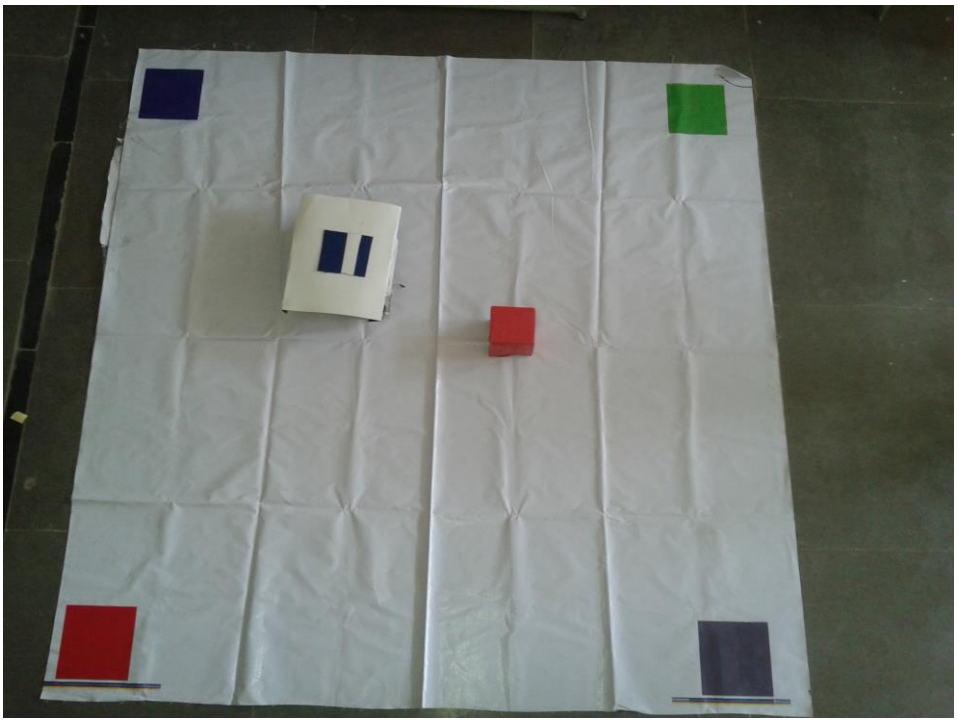


figure 10.6

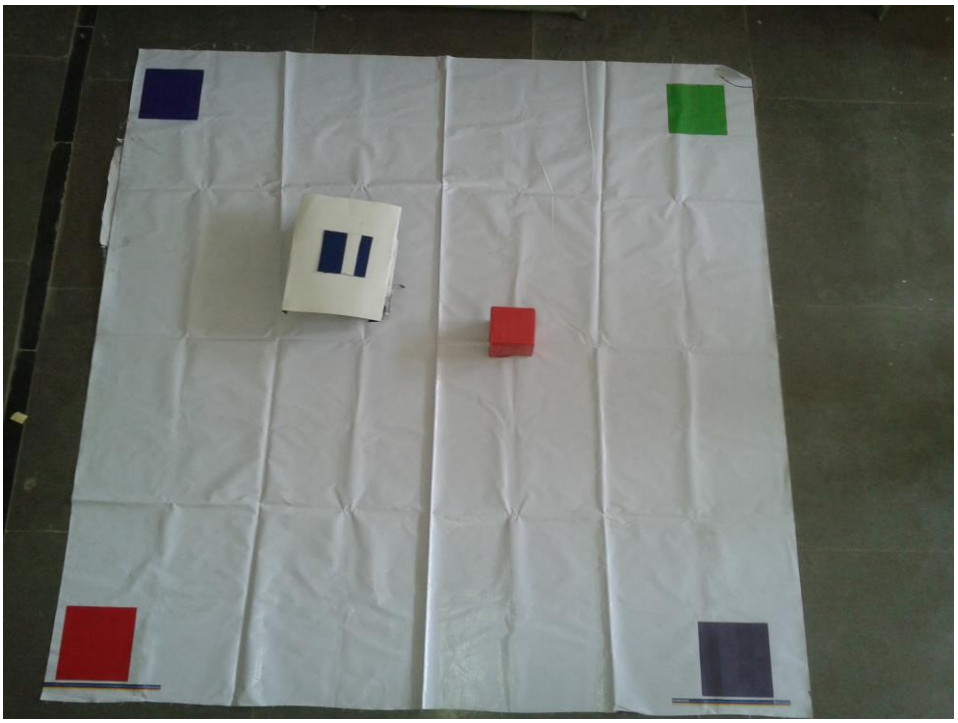


figure 10.7 Robot with arena

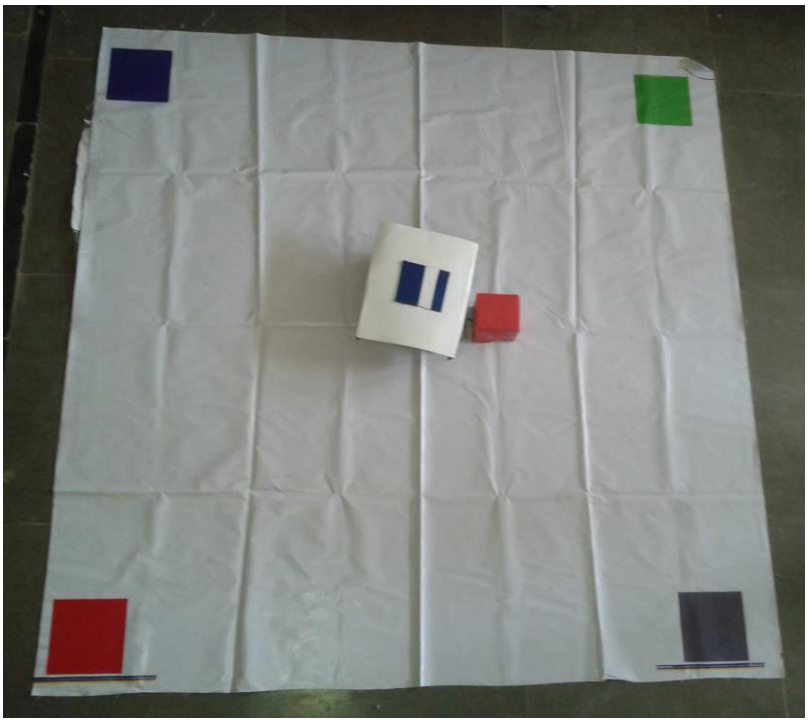


figure 10.8 robot at the site of the object

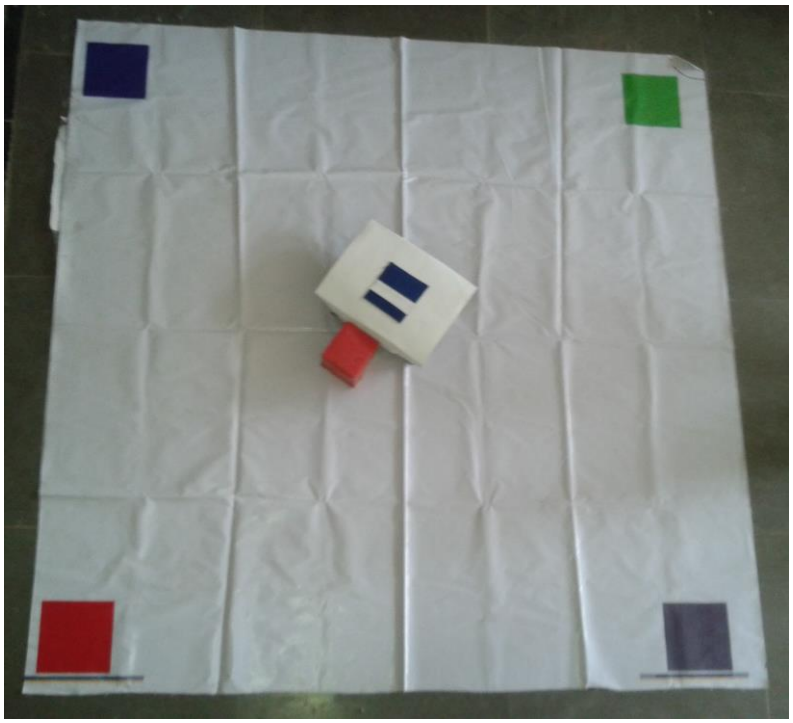


Figure 10.9 robot aligning object towards destination

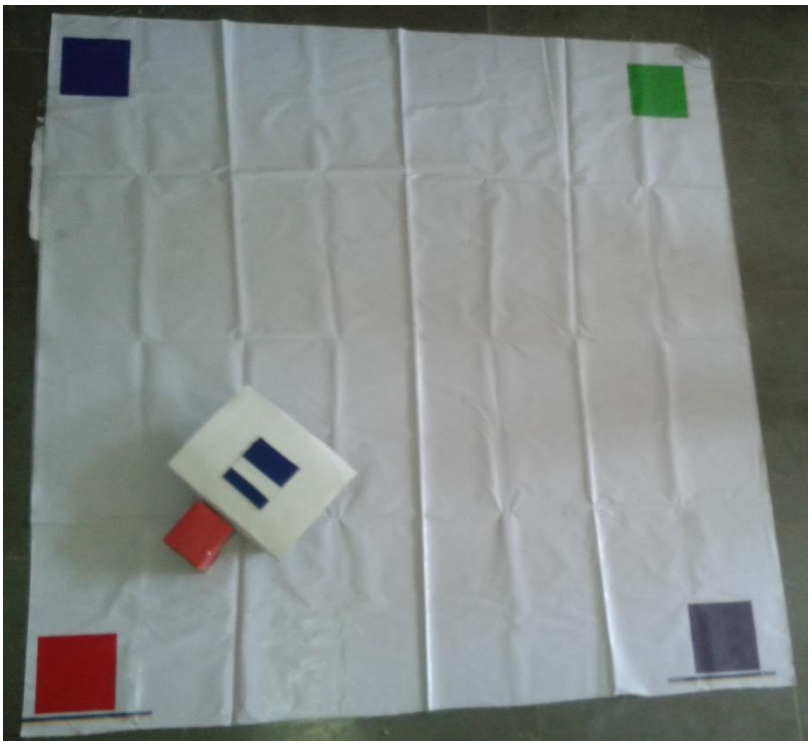


figure 10.10 robot on the way to its destination

10.5 SOURCE CODE

```
clear all;
close all;
imtool close all;
clc;


s = serial('COM5','BAUD',4800);
fopen(s);

se=strel('disk',6);
se1=strel('disk',5);

vid=videoinput('winvideo',1,'YUY2_640x480');
vid.ReturnedColorspace = 'rgb';
preview(vid);
pause(2);

while(1)

   im=getsnapshot(vid);

   % imtool(im);

   R = im(:,:,1);
   G = im(:,:,2);
   B = im(:,:,3);

   rth = graythresh(R);
   gth = graythresh(G);
   bth = graythresh(B);
```

```
red = im2bw(R,95/256);
green = im2bw(G,gth);
blue = im2bw(B,40/256);

R_only=red-green-blue;
G_only=green-blue-red;
B_only=blue-red-green;

sky = blue+green-red;

R_only = im2bw(R_only,0);
G_only = im2bw(G_only,0);
B_only = im2bw(B_only,0);

sky = im2bw(sky,1);

R_only=imclose(R_only,se);
G_only=imclose(G_only,se);
B_only=imclose(B_only,se);

sky = imclose(sky , se);

R_only=imopen(R_only,se1);
G_only=imopen(G_only,se1);
B_only=imopen(B_only,se1);

sky = imopen(sky , se1);

% imtool(R_only);
% imtool(G_only);
% imtool(B_only);
% imtool(sky);

%%
 L = bwlabel(R_only);
```

```
if(max(max(L)) == 2)
    image = R_only;
else
    L = bwlabel(G_only);
    if(max(max(L)) == 2)
        image = G_only;
    else
        L = bwlabel(sky);
        if (max(max(L)) == 2)
            image = sky;
        else
            fwrite(s,'s');   %% task is complete
            clear image;
        end
    end
end
o_d = regionprops(image,'Area','Centroid');
```

```
    if(o_d(1).Area > o_d(2).Area)

        destination = o_d(1).Centroid;

        object = o_d(2).Cenroid;

    else if (o_d(1).Area < o_d(2).Area)

        destination = o_d(2).Centroid;

        object = o_d(1).Cenroid;

      end

    end


    [Bb,Lb] = bwboundaries(B_only,'noholes');
    bot = regionprops(B_only,'Area','Centroid');

    r_cent = ((bot(1).Area * bot(1).Centroid)+(bot(2).Area
*bot(2).Centroid))/((bot(1).Area + bot(2).Area));

    if(bot(1).Area>bot(2).Area)

      head = (bot(2).Centroid);

      tail = (bot(1).Centroid);

    else if (bot(2).Area>bot(1).Area)

      head = (bot(1).Centroid);

      tail = (bot(2).Centroid);

      end
```

```
end
 ht = head - tail;
% robot angle
if(t(x)>0 && ht(y)>0)
   theta_r = atan(ht(x)/ht(y));
else if (ht(x)>0 && ht(y)<0)
      theta_r = (pi/2) + atan(-ht(y)/ht(x));
   else if (ht(x)<0 && ht(y)<0)
         theta_r = (pi) + atan(-ht(x)/ht(y));
      else if (ht(x)<0 && ht(y)>0)
            theta_r = (3*pi/2) + atan(-ht(y)/ht(x));
         end
      end
   end
end
ot = object - tail;
```

```
% object robot angle
if(ot(x)>0 && ot(y)>0)
   theta_o = atan(ot(x)/ot(y));
else if (ot(x)>0 && ot(y)<0)
      theta_o = (pi/2) + atan(-ot(y)/ot(x));
   else if (ot(x)<0 && ot(y)<0)
         theta_o = (pi) + atan(-ot(x)/ot(y));
      else if (ot(x)<0 && ot(y)>0)
            theta_o = (3*pi/2) + atan(-ot(y)/ot(x));
         end
      end
   end
end
ob_angle = (theta_o - theta_r)*180/pi;
oh_dist = sqrt((object(1)-head(1))^2 + (object(2)-head(2))^2);
dr = destination - r_cent;
```

```
% path angle
if(dr(x)>0 && dr(y)>0)
   theta_p = atan(dr(x)/dr(y));
else if (dr(x)>0 && dr(y)<0)
      theta_p = (pi/2) + atan(-dr(y)/dr(x));
   else if (ht(x)<0 && ht(y)<0)
         theta_p = (pi) + atan(-dr(x)/dr(y));
      else if (dr(x)<0 && dr(y)>0)
            theta_p = (3*pi/2) + atan(-dr(y)/dr(x));
         end
      end
   end
end
angle = (theta_r - theta_p)*180/pi;
```

```
if (oh_dist <= 75)

   if ((angle < 10) && (angle > -10))

   fwrite(s,'f');
   fwrite(s,'s');

   else if (angle < 0)

      fwrite(s,'l');
      fwrite(s,'s');

      else if (angle > 0)

         fwrite(s,'r');
         fwrite(s,'s');

         end
      end
   end


else

   if(ob_angle > -5 && ob_angle < 5)

      fwrite(s,'f');
      fwrite(s,'s');

   else if (ob_angle < 0)

         fwrite(s,'r');
         fwrite(s,'s');

      else if (ob_angle > 0)
```

```
                fwrite(s,'l');
                fwrite(s,'s');

            end
        end
    end
end


clear o_d;
clear bot;
clear head;
clear ht;
clear ot;
clear tail;
clear dr;
clear theta_r;
clear theta_o;
clear theta_p;
clear angle;

end

clear vid;
```