1. INTRODUCTION

With an objective of involving both software as well as hardware aspect in our project, we chose to make a track solving robot as our minor project and named it "Vehicle guidance by image processing". In this automated era, machines are becoming more reflexive day by day. Human intervention on machines is being lessened gradually. Complying the same principle, this project aims to demonstrate a system comprising of a path solver (working as a "brain") and a hardware that accepts data solved by the solver, which in this case is a laptop, and moves accordingly.

1.1 Problem Definition

Automated industry and electronics has been one of the pace catching subject matter in the world of science. Any electronic system, if is automated, is considered to be a success. However, in the field of transportation, devices have not been able to be reflexive as they are supposed to. Hence, taking this topic into account, we attempted to make path-movement self-referent.

Furthermore, the maze solving algorithm implemented in this project is efficacious. This algorithm might help in solving complex puzzles within a fraction of a time with almost entire accuracy.

1.2 Scope and Applications

This kind of system has not yet been, but can be implemented in delivering packages on online shopping. At present, drones have been used for delivering purpose, which is an effective way as compared to human delivering system, but is not the best possible way. Applying the principle we used in our project, multiple packages can be delivered to nearby destinations using different roadway robots controlled by a single drone. Furthermore, the path solving algorithm has numerous applications, extending from transit node routing to Rubik's cube solver (edge solving).

This kind of system can also be used in giga-factories where there is a long route for assembling the components. Machines can be embedded with the software so that they could detect the presence of components and assemble them accordingly.

1.3 Goals and Objectives

The main goal of this project is to acquire knowledge regarding communication between software and hardware. Many aspects of software including python, OpenCV, Bluetooth communication and on the hardware part, AVR microcontroller, Bluetooth module, motor driver IC and other minor ironware were used. This system can be defined as an input driven system whose output is the movement of a robotic car.
Some other objectives are listed below:

- To articulate hardware with software.
- To design a fully automated system.
- To learn serial communication (UART) in microcontroller.
- To design a system that can solve path finding problems.
- To promote effective package delivery system.

2. LITERATURE REVIEW

This project can simply be taken as a person driving a vehicle, who knows the entire path from start to finish. Here, image processing acts as a person whereas the robotic car is the vehicle. Implementing the communication between software and hardware had been an interesting subject though. During the progress of this project, we came to divulge the unsung importance of image processing and its applications.

According to Elon Musk, CEO of Tesla Motors, self-autonomous cars will have hundred percent safeties by 2030 A.D. This proves the growing market of autonomous transportation. Having no other exceptions, this project also deals with the same subject, but with different principle.

One of the major drawbacks of existing self-autonomous vehicles is that, it has the software build-in in itself. Hence, it can only process the images it can see. This creates a major drawback of fortuity if sudden changes occur. This project eradicates the existing problem by pre-determining the whole path and making necessary calculations before the vehicle reaches to the predetermined point.

3. SOFTWARE AND HARDWARE OVERVIEW

OpenCV (Open Source Computer Vision) library has been used on top of python programming language. Left line following algorithm has been implemented with some minor tweaks. This program first detects the path drawn on the flex over which the car will move and then solves it to find a path from start to finish (which is identified by different colors placed on start and finish point). Real time position of the robotic car is also incurred via color detection which enables the system to send data to the robot periodically regarding its movement and turns.

3.1 Software

3.1.1 Python 2.7 :

Python is an object-oriented, high-level programming language. It is simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. It has high level build in data structures with dynamic typing. Python supports modules and packages, which encourages program modularity and code reuse.

For this project, python 2.7 was used since it is compatible with OpenCV3 and its tutorials and examples are comfortably encountered on internet.

3.1.2 OpenCV 3.0:

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. OpenCV supports a wide variety of programming languages like C++, Python, Java and is available on different platforms including Windows, Linux, OS X, Android, iOS etc. For this project, many features of OpenCV including color detection, coordinate manipulation, real time operation, line detecting algorithm, left line following algorithm has been implemented.

3.1.3 OpenCV- Python:

OpenCV-Python works on a Python wrapper around original C++ implementation. The support of Numpy makes the task more easier. Numpy is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations we can do in Numpy, we can combine it with OpenCV, which increases number of weapons in our arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this

3.2 HARDWARES

3.2.1 AVR Atmega-8:

Atmel 8-bit AVR is a RISC-based microcontroller that incorporates 8KB of programmable flash memory, 1KB of SRAM, 512B EEPROM, and a 6 or 8 channel 10-bit A/D converters. The device supports throughput of 16 MIPS at 16 MHz and operates between 2.7-5.5 volts.

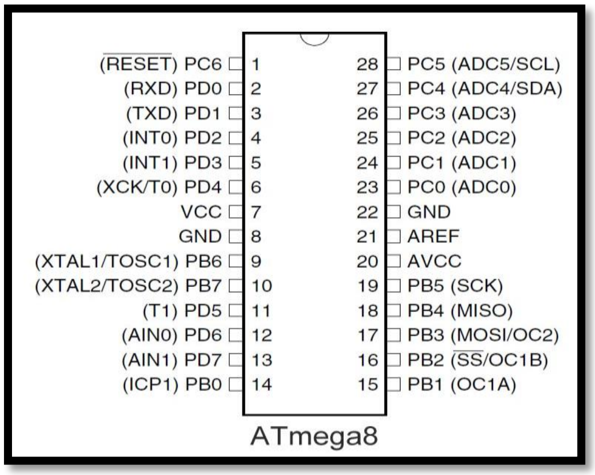


Figure 3.1

PINC0- PINC3 are connected to the input pins of L293D motor driver IC. Internal clock of the microcontroller has been used which has a clock frequency of 3MHz. It with powered with the same battery which is used to power the two motors, and the motor driver IC.

3.2.2 L293D Motor Controller IC:

L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

There are two Enable pins on l293d. Pin 1 and pin 9, for being able to drive the motor, the pin 1 and 9 need to be high. For driving the motor with left H-bridge you need to enable pin 1 to high. And for right H-Bridge you need to make the pin 9 to high. If anyone of the either pin1 or pin9 goes low then the motor in the corresponding section will suspend working. It’s like a switch.

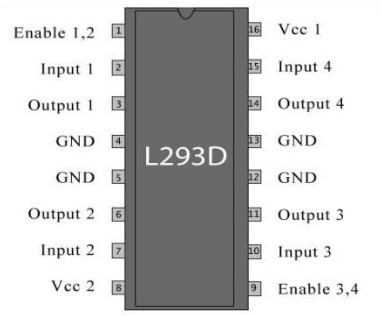


Figure 3.2

3.2.3 HC-05 Bluetooth Module:

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Some of its features are low Power 1.8V operation ,1.8 to 3.6V I/O, UART interface with programmable baud rate, integrated antenna, master and slave configuration and so on.

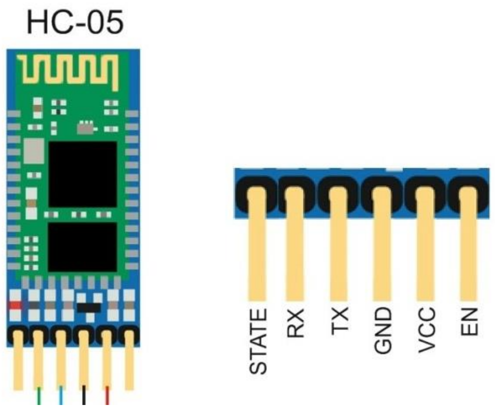


Figure 3.3

3.2.4 LM7805 Voltage Regulator:

7805 is a voltage regulator integrated circuit. It is a member of 78xx series of fixed linear voltage regulator ICs. The voltage source in a circuit may have fluctuations and would not give the fixed voltage output. The voltage regulator IC maintains the output voltage at a constant value. The xx in 78xx indicates the fixed output voltage it is designed to provide. 7805 provides +5V regulated power supply. Capacitors of suitable values can be connected at input and output pins depending upon the respective voltage levels.

This was used in order to supply power to AVR and L293D since they both work on 5V supply.

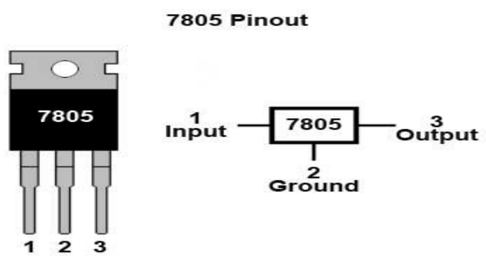
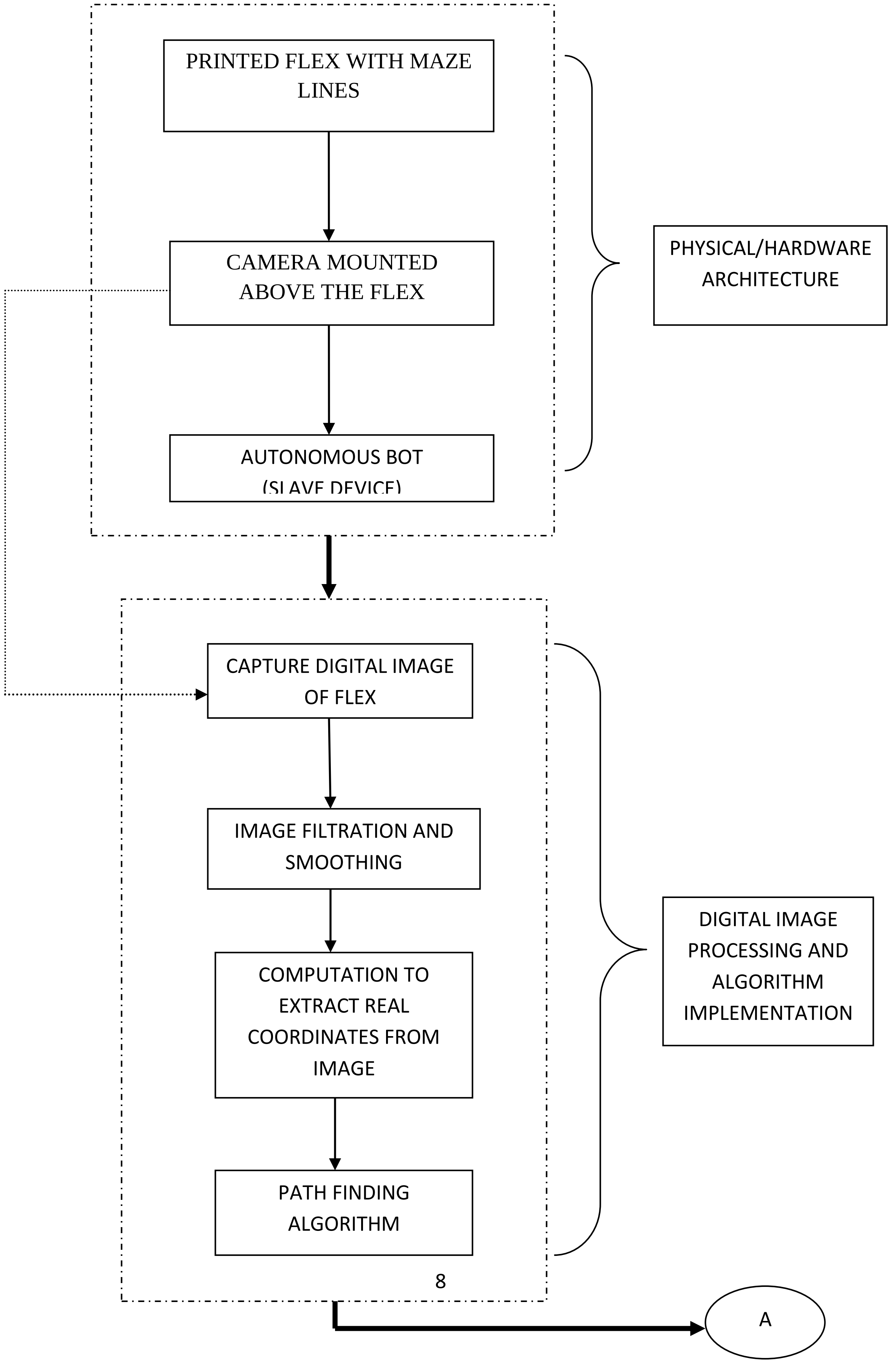
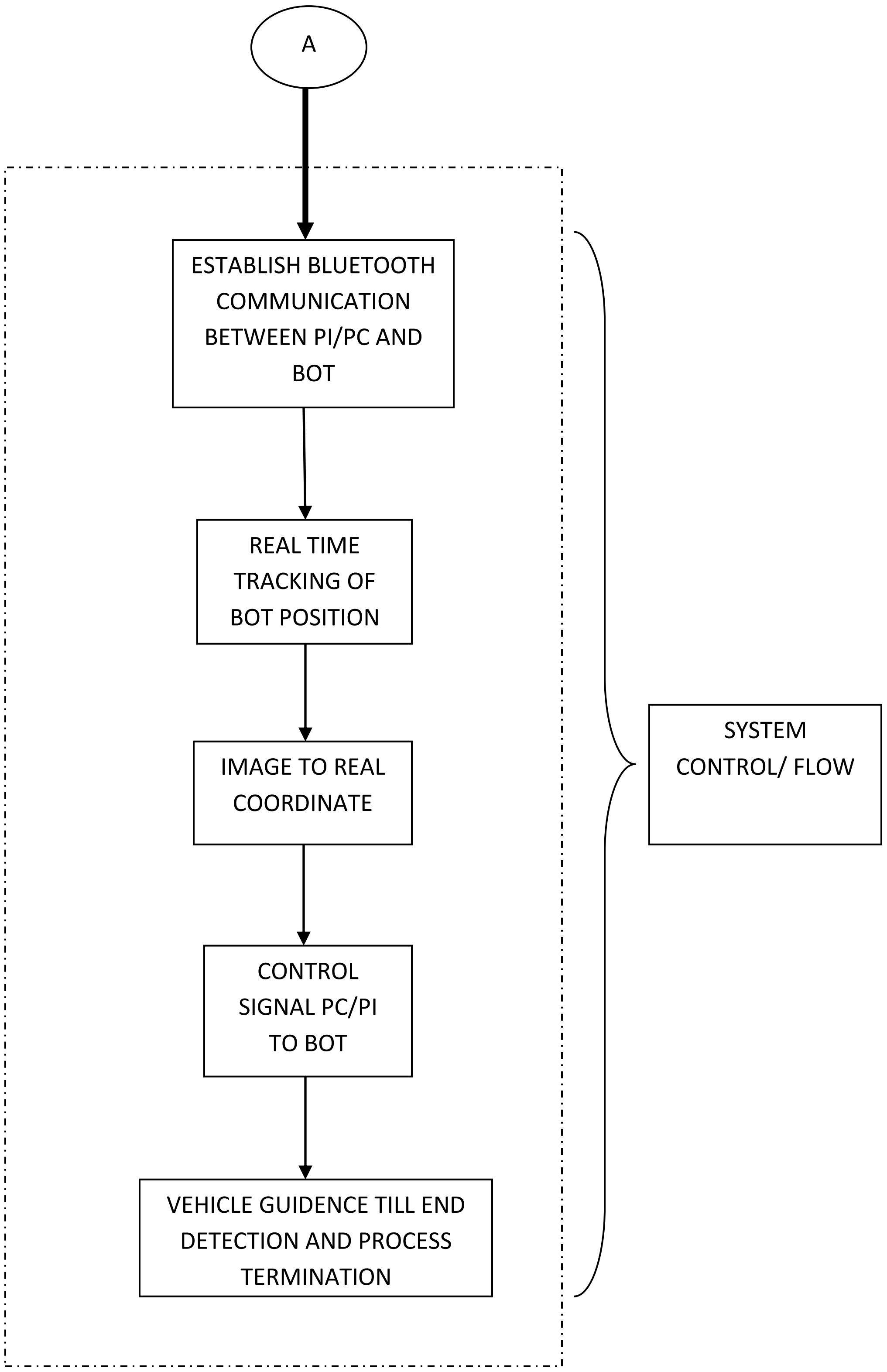


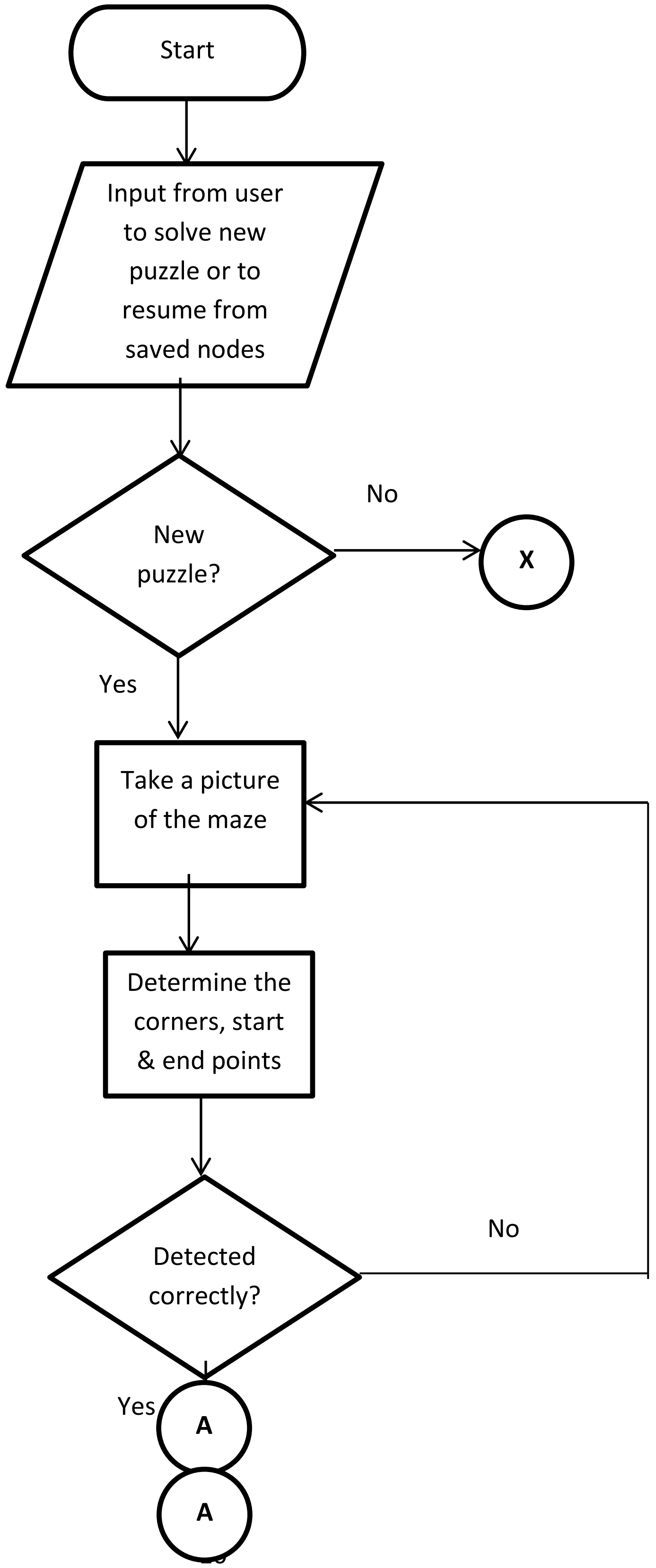
Figure 3.4

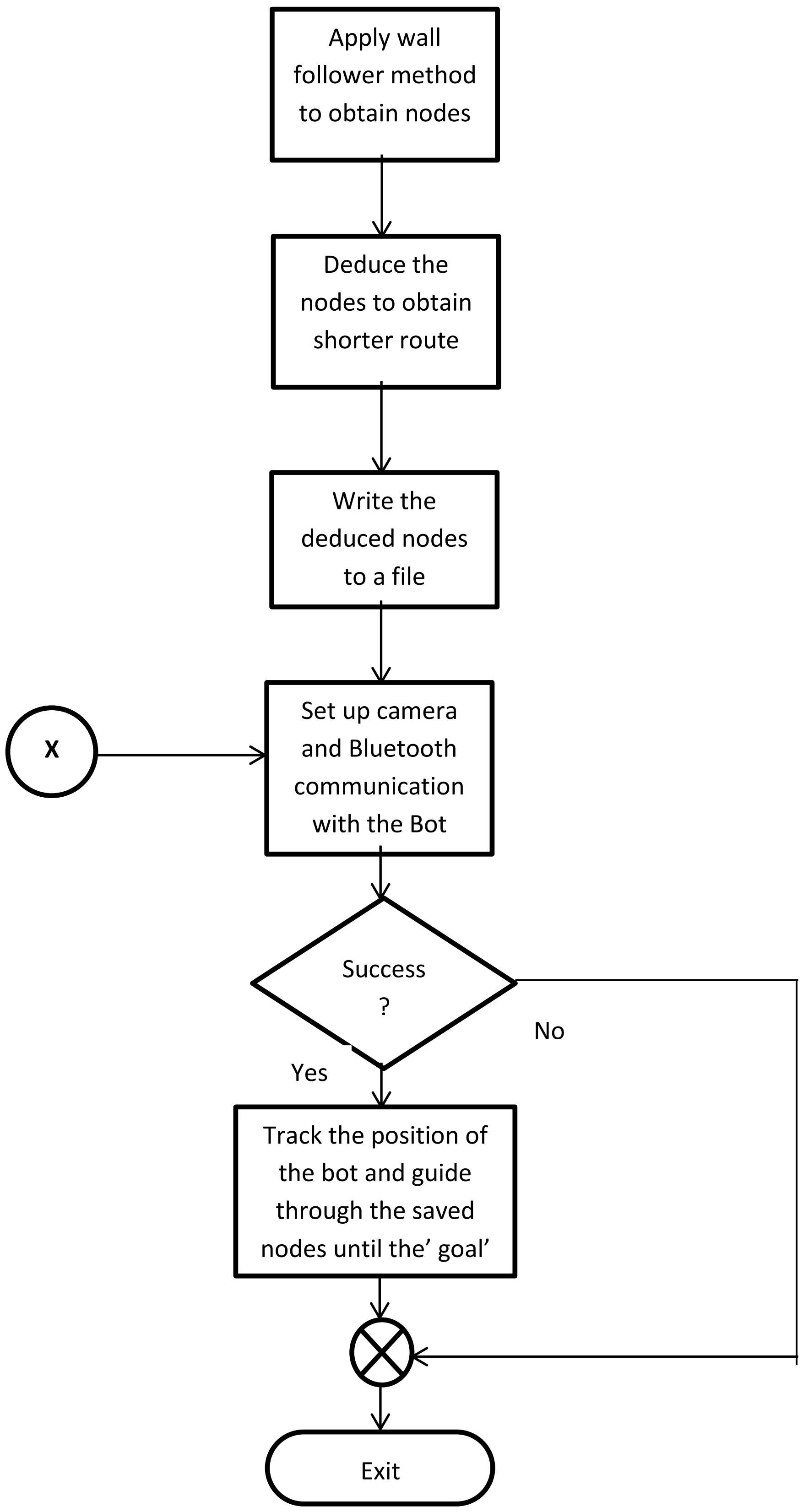
4. SYSTEM DESIGN AND ARCHITECTURE


PRINTED FLEX WITH MAZE LINES
CAMERA MOUNTED ABOVE THE FLEX
AUTONOMOUS BOT (SLAVE DEVICE)
PHYSICAL/HARDWARE ARCHITECTURE
CAPTURE DIGITAL IMAGE OF FLEX
IMAGE FILTRATION AND SMOOTHING
COMPUTATION TO EXTRACT REAL COORDINATES FROM IMAGE
PATH FINDING ALGORITHM
DIGITAL IMAGE PROCESSING AND ALGORITHM IMPLEMENTATION
A


A
ESTABLISH BLUETOOTH COMMUNICATION BETWEEN PI/PC AND BOT
REAL TIME TRACKING OF BOT POSITION
IMAGE TO REAL COORDINATE
CONTROL SIGNAL PC/PI TO BOT
VEHICLE GUIDENCE TILL END DETECTION AND PROCESS TERMINATION
SYSTEM CONTROL/ FLOW

Flowchart:


Start
Input from user to solve new puzzle or to resume from saved nodes
New puzzle?
No
X
Yes
Take a picture of the maze
Determine the corners, start & end points
Detected correctly?
No
Yes
A
A


Apply wall follower method to obtain nodes
Deduce the nodes to obtain shorter route
Write the deduced nodes to a file
X
Set up camera and Bluetooth communication with the Bot
Success ?
No
Yes
Track the position of the bot and guide through the saved nodes until the' goal'
Exit

5. Methodology

5.1 Object Detection Based on Color Detection with contour Formation

In this approach, we have used the object detection technique based on the individual color identification. For that, it is very important to change image to HSV format such that each color could be filtered out using proper threshold values. Figure 5.1 shows the conversion from Grayscale to HSV. OpenCV function *cvtColor(src, hsv, CV_BGR2HSV)* is used to convert RBG to HSV.

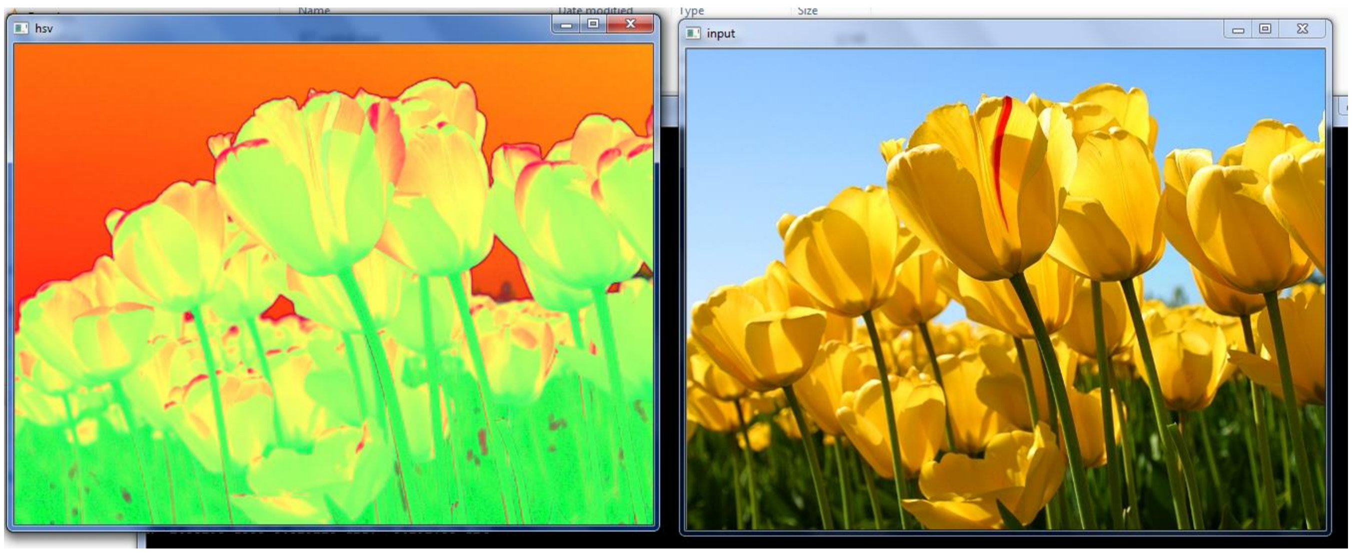


Figure 5.1

After proper threshold, individual color could be separated. Then, it is necessary to setup a boundary of that color (contouring). Contours can be explained simply as a curve joining all the continuous points(along the boundary),having same color or intensity.

The contours are a useful tool for shape analysis and object detection and recognition. For better accuracy, binary images could be used. In OpenCV, finding contours is like finding white object from black background so, object to be found should be white and background should be black. OpenCv function cv2.findContours() is used to find the contour which has 3 arguments. First one is source image, second is contour retrieval mode, third is contour approximation method. The function returns variable with the coordinates value in (x, y) array. Contour approximation method is implemented according to the requirement of coordinates.(whether we need coordinates of the whole boundary or corner points). In our case, we have used *cv2.CHAIN_APPROX_SIMPLE* as the argument that returns only 4 corner points. (Sufficient enough for our computation, figure 5.2).

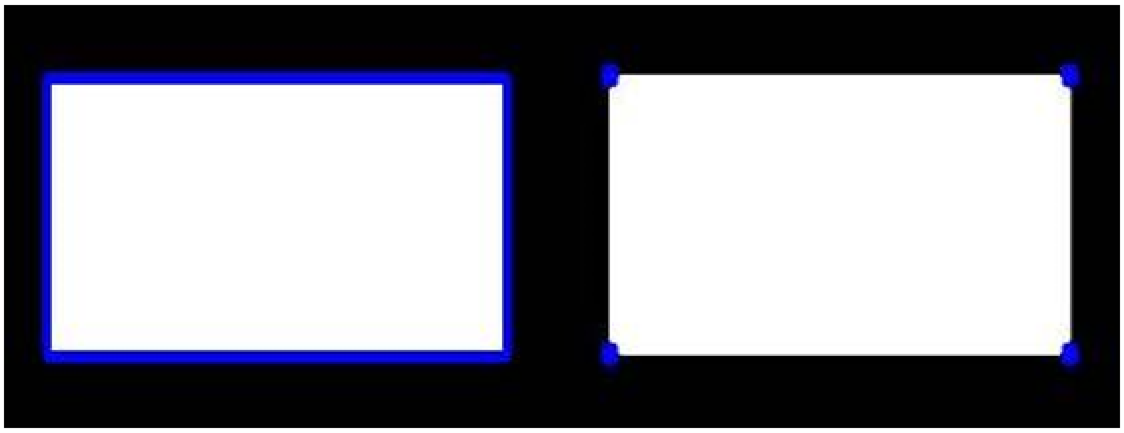


Figure 5.2

Figure 8: Contour approximation methods

In this way, the real coordinates of the required object has been obtained which is later used for further computation.

5.2 Image Smoothing and Morphological Operations

Prior to any further digital processing of the image, the noise present in the image should be reduced. For that, smoothing- also called as blurring has been used. To perform a smoothing operation we have applied a filter to our image. The most common type of filters are linear, in which an output pixel's value (i.e. g *(i, j)*) is determined as a weighted sum of input pixel values (i.e. f *(i + k, j + l)*):

$$g(i, j) = \sum k, l\ f(i + k, j + l) * h(k, l)$$

h *(k,l)* is called the kernel, which is nothing more than the coefficients of the filter.

It helps to visualize a filter as a window of coefficients sliding across the image. The most useful filter (although not the fastest) is Gaussian filter and it has been used in our case. Gaussian filtering is done by convolving each point in the input array with a Gaussian kernel and then summing them all to produce the output array.A 2D Gaussian can be represented as:

$$G_0(x, y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}}$$

where μ is the mean (the peak) and σ represents the variance (per each of the variables x and y).

To perform the Gaussian filtering, we have used opencv function:

GaussianBlur(src, dst, Size(i, i), 0, 0);

• *src:* Source image

• *dst*: Destination image

• *Size(w, h)*: The size of the kernel to be used (the neighbors to be considered). w and h have to be odd and positive numbers otherwise thi size will be calculated using the σx and σy arguments.

• *σx*: The standard deviation in x. Writing 0 implies that σx is calculated using kernel size.

• *σy*: The standard deviation in y. Writing 0 implies that σy is calculated using kernel size.

Morphological operation**:** A set of operations that process images based on shapes. Morphological operations apply a structuring element to an input image and generate an output image. Basically there are two types: Erosion and Dilation. It has been used for the following purpose:

– Removing noise.

– Isolation of individual elements and joining disparate elements in an image.

– Finding of intensity bumps or holes in an image.

Dilation: This operation consists of convoluting an image A with some kernel (B), which can have any shape or size, usually a square or circle. The kernel B has a defined anchor point, usually being the center of the kernel. As the kernel B is scanned over the image, we compute the maximal pixel value over lapped by B and replace the image pixel in the anchor point position with that maximal value. It could be seen that this maximizing operation causes bright regions within an image to "grow" (therefore the name dilation).

Erosion: This operation is the sister of dilation. What this does is to compute a local minimum over the area of the kernel. As the kernel B is scanned over the image, we compute the minimal pixel value overlapped by B and replace the image pixel under the anchor point with that minimal value.

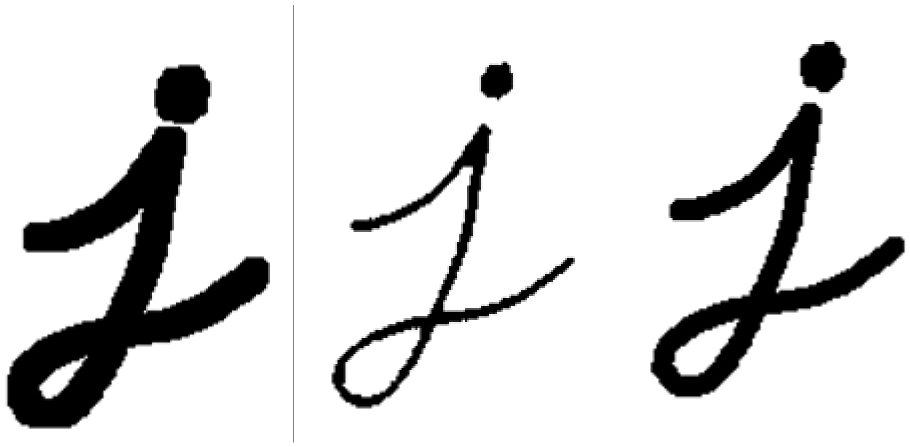


Figure 5.3

Opening: It is obtained by the erosion of an image followed by dilation.

dst = open(src,element) = dilate(erode(src,element))

Closing: It is obtained by the dilation of an image followed by erosion.

dst = close(src,element) = erode(dilate(src,element))

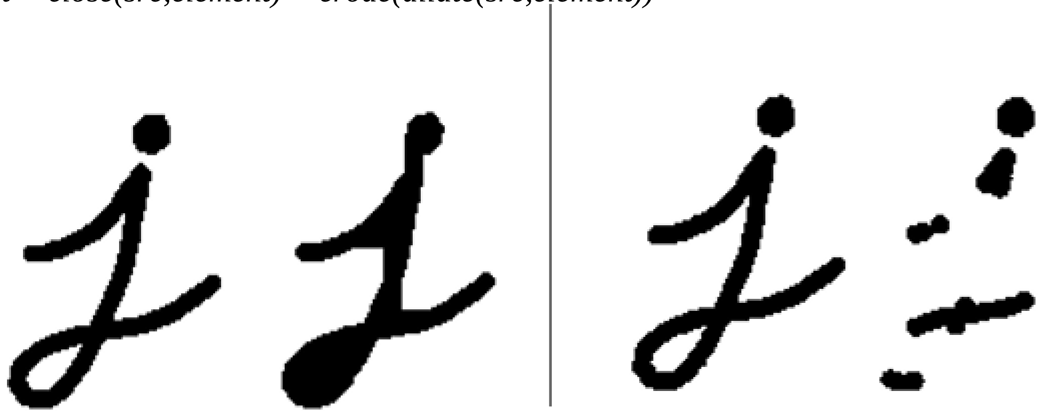


Figure 5.4

5.3 Line Detection

Detection of lines from the captured image of the maze is the foremost step to be computed before any algorithms for path calculation could be applied. Moreover, the maze could only be visualized for further computation only after the real co-coordinates of the actual maze lines are obtained from the digital image. For this an OpenCV class "cv::LineSegmentDetector" is implemented whose computation is based on explanation provided in the IPOL Journal article *LSD: a Line Segment Detector.*

LSD is a linear-time Line Segment Detector giving subpixel accurate results. It is designed to work on any digital image without parameter tuning. It controls its own number of false detections: on average, one false alarm is allowed per image. The method is based on *Burns, Hanson, and Riseman's method* , and uses an *a contrario* validation approach according to *Desolneux, Moisan, and Morel's theory*.

LSD is aimed at detecting locally straight contours on images, called line segments. Contours are zones of the image where the gray level is changing fast enough from dark to light or the opposite. Thus, the gradient and level-lines of the image are key concepts and are illustrated in figure 1. The algorithm starts by computing the level-line angle at each pixel to produce a level-line field, i.e., a unit vector field such that all vectors are tangent to the level line going through their base point. Then, this field is segmented into connected regions of pixels that share the same level-line angle up to a certain tolerance τ . These connected regions are called line support regions, see figure 5.4.

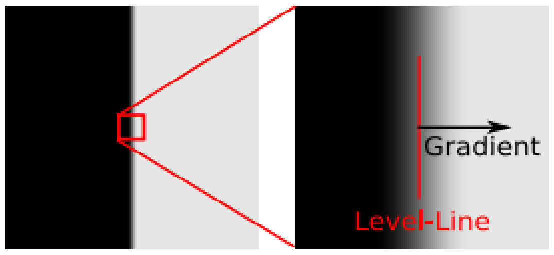


Figure 5.4

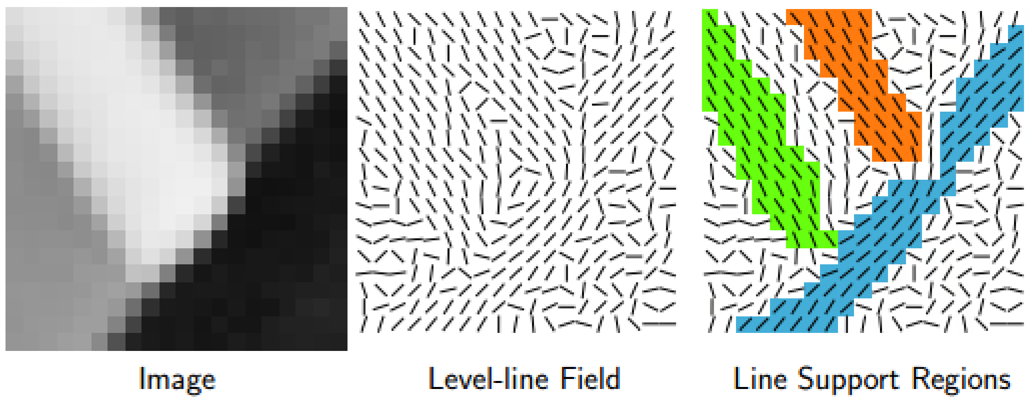


Figure 5.5

Each line support region (a set of pixels) is a candidate for a line segment. The corresponding geometrical object (a rectangle in this case) must be associated with it. The principal inertial axis of the line support region is used as main rectangle direction; the size of the rectangle is chosen to cover the full region . Each rectangle is subject to a validation procedure. The pixels in the rectangle whose level-line angle corresponds to the angle of the rectangle up to a tolerance τ are called aligned points . The total number of pixels in the rectangle, n, and its number of aligned points, k, are counted and used to validate or not the rectangle as a detected line segment. The validation step is based on the *a contrario* approach and the Helmholtz principle proposed by *Desolneux, Moisan, and Morel.*

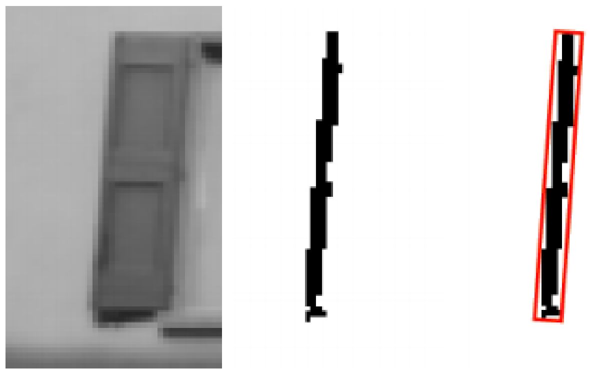


Figure 5.6

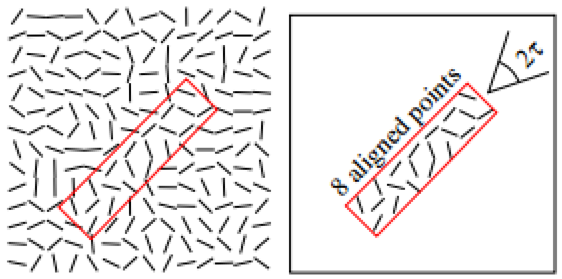


Figure 5.7

5.4 Solving the puzzle

After detecting the lines in the puzzle, we are provided with end points of each line in Cartesian system. The location of the starting and ending point for the vehicle is previously detected. The detection of path starts from the starting point. Initially, 'Wall Follower Algorithm' is applied to determine the route form 'start' to 'end' of the puzzle.

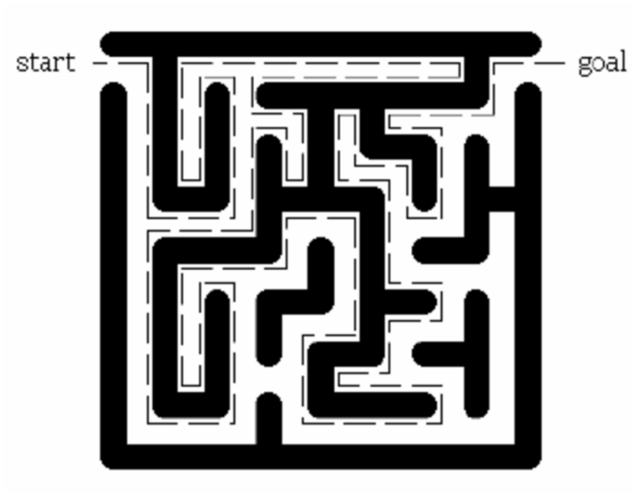


Figure 5.8

In above fig, we can see that the 'goal' can be reached from the 'start' by just following the wall in the left (or right). Although this method is foolproof for solving maze type puzzles, it does not determine the shortest path.

So to determine the shorter route in our project we have to improvise this algorithm. Now all the turning points (nodes) are determined by wall-follower algorithm, we can represent the maze as tree.

As shown in the figure below, each letter represents a node (turning point in the maze). We have already achieved Cartesian coordinate of each node. Now 'dead-end' nodes (i.e., nodes – 'D', 'J', 'I', 'L' and 'O') can be avoided which leads to our 'goal'.

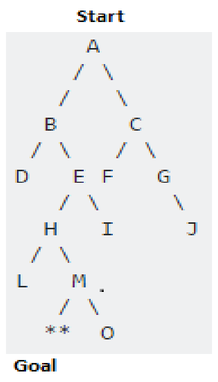


Figure 5.9

In this way a shorter path is determined and the maze is solved.

5.5 Bluetooth communication (computer/raspberry pi)

Raspberry pi/ computer has built in Bluetooth hardware module. However, to access it, two python modules bluez and python-serial are required. For Bluetooth communication between HC05 and master device(computer/raspberry pi), it is important to set up another protocol over the Bluetooth, which is named RFCOMM to emulate the serial connection between these two devices. The protocol setup could be done by modifying the "rfcomm.conf" file and binding these two devices in one channel which could be simply done by the command sudo rfcomm.config *hci0 ******** 1* where ******** is the device address(HC05) and both devices are bound in channel 1

To communicate, a common baud rate should be set up and bluetoothSerial.Write and bluetoothSerial.Read are to be used to write character to the port of microcontroller via Bluetooth module and to read it from there. Communication remains unless interrupted. In our case, only characters are written to microcontroller rather than string.

In the bot side we have used HC-05 controlled by AVR microcontroller. The communication between the microncontroller and avr is asynchronous serial communication.

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection

• Framing Error Detection
• Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
• Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete
• Multi-processor Communication Mode
• Double Speed Asynchronous Communication Mode

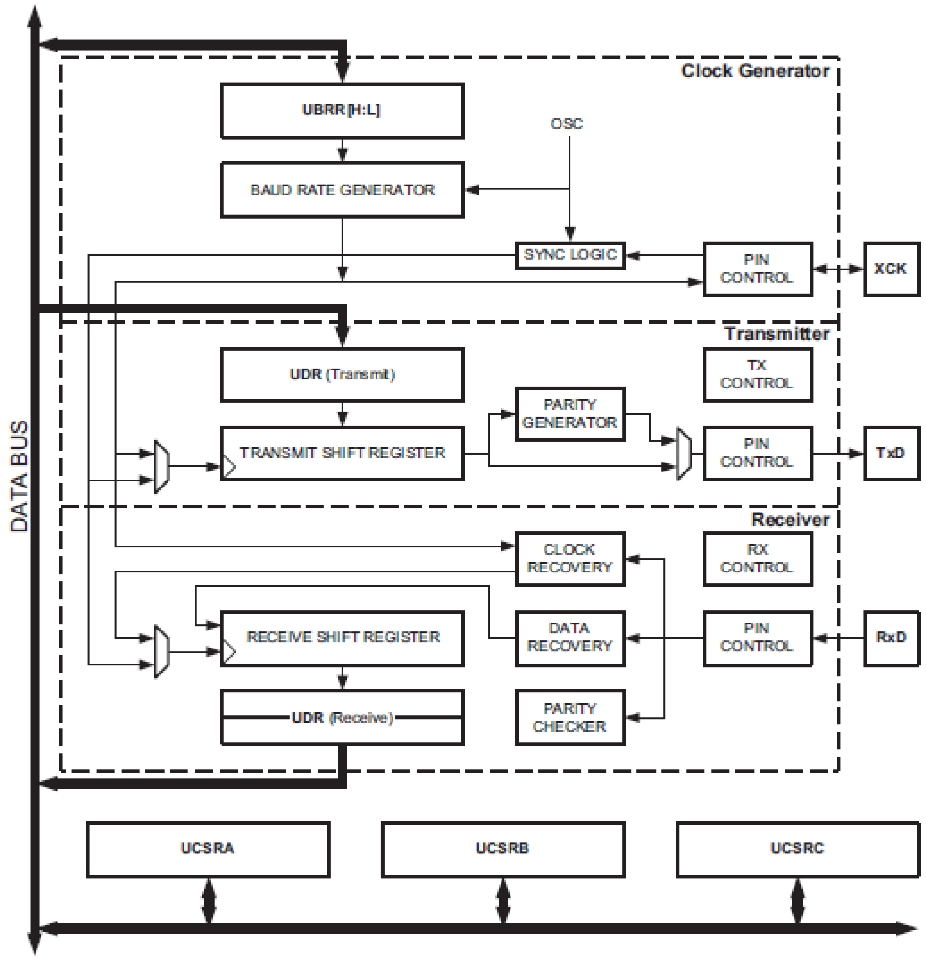


Figure 5.9

The dashed boxes in the block diagram separate the three main parts of the USART : Clock Generator, Transmitter and Receiver. Control Registers are shared by all units.The clock generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCK (Transfer Clock) pin is only used by Synchronous Transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, parity generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery-units. The recovery units are used for asynchronous data reception. In addition to the recovery-units, the receiver includes a parity checker, control logic, a Shift Register and a two level receive buffer (UDR). The receiver supports the same frame formats as the transmitter, and can detect frame error, data overrun and parity errors.

Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock (fosc), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= fosc/(UBRR+1)). The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSEL, U2X and DDR_XCK bits.

For asynchronous normal mode,

BAUD = $f_{osc}/16(UBRR+1)$

For synchronous master mode,

BAUD = $f_{osc}/2(UBRR+1)$

Where f_{osc} refers system oscillator clock frequency

Frame Format

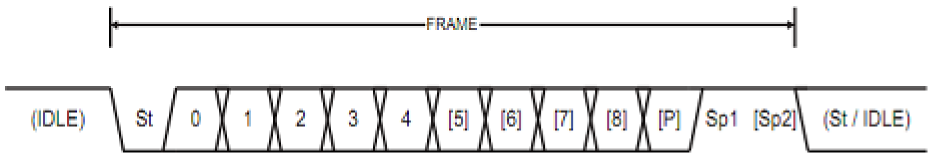


Figure 5.10

The frame format used by the USART is set by the UCSZ2:0, UPM1:0, and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter. The USART Character SiZe (UCSZ2:0) bits select the number of data bits in the frame. The USART Parity mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBS) bit. The receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

5.6 PCB DESIGN PROCESS

PCB design process was concluded implementing the following steps:

1. Schematic was made using PCB Wizard was printed in a glossy sheet of paper.
2. A single sided PCB board was used, after it was oxidized, where the schematic was imprinted.
3. Etching process was completed by drowning thus imprinted PCB board in an aqueous solution of heated $FeCl_3$.
4. The imprinted schematic was scrubbed by steel wool to disclose the copper path.
5. Nodes were drilled using 1 mm drill bit and hence respective components were soldered.

The PCB schematic and circuit diagram can be found in appendix section of this report.

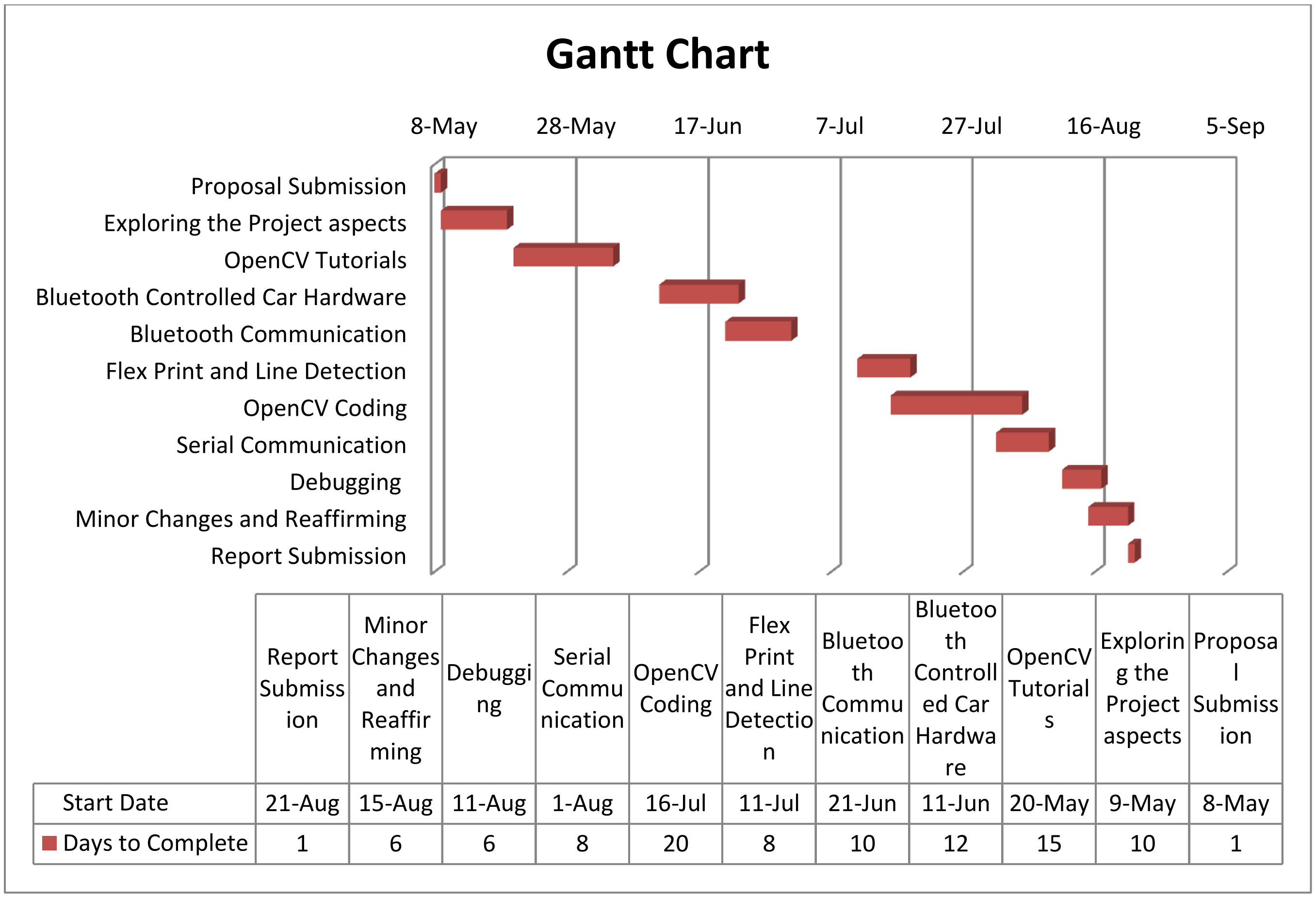
6. PROJECT MANAGEMENT

6.1 Cost Management

S.N	Components	Cost
1.	AVR Atmega-8	Rs. 250
2.	L293D-Motor IC	Rs. 100
3.	HC-05 Bluetooth Module	Rs. 900
4.	Chasis	Rs. 250
5.	Flex	Rs. 400
6.	Camera	Rs. 300
7.	A Pair of Motors	Rs. 300
8.	A Pair of Wheels	Rs. 180
9.	Castor Ball	Rs. 80
10.	Copper Board (PCB)	Rs. 250
11.	FeCl3 (For Etching)	Rs. 250
12.	Power Source (Battery)	Rs. 300
13.	7805 voltage controller	Rs. 40
		Total- Rs

Table 6.1

6.2 Time Management



	Report Submission	Minor Changes and Reaffirming	Debugging	Serial Communication	OpenCV Coding	Flex Print and Line Detection	Bluetooth Communication	Bluetooth Controlled Car Hardware	OpenCV Tutorials	Exploring the Project aspects	Proposal Submission
Start Date	21-Aug	15-Aug	11-Aug	1-Aug	16-Jul	11-Jul	21-Jun	11-Jun	20-May	9-May	8-May
🟥 Days to Complete	1	6	6	8	20	8	10	12	15	10	1

Table 6.2

7. RESULT AND ANALYSIS

As soon as the image from the flex is captured by the camera, it calculates the nodes of the path implementing line detecting algorithm and hence stores it in a text file. With the help of color detection principle, start and end points are determined. Now the system starts to compute the path from start to finish using left line following algorithm with some minor adjustments. This takes about 10 seconds and hence stores the coordinates in a text file.

The vehicle, after it is placed on the start point, the system detects it via the color placed on top of it. Hence the real time coordinate of the vehicle is computed and is compares with the previously calculated path and makes necessary decision regarding the movement of the vehicle. The vehicle takes nearly 20-25 seconds to reach from starting to end point, provided that they are same. The start and end points can be altered, resulting in a new path computation and hence the time taken changes accordingly.

The ten second time taken by the system to calculate the path is due to the fact that it tries to avoid walls on each and every point during its path calculation. The camera used produces perspective vision which causes error in the detection of the bot. To determine the exact location of the bot, the captured image has to be converted to orthographic projection.

8. CONCLUSION AND FUTURE ENHANCEMENT

This minor project on Image processing will prove boon for the engineering students of IOE who are interested in the same field. Via this project, we the engineering students are able to tackle with sophisticated instruments while designing PCB. This project is beneficial for students as it will help to compete with ever-growing challenges and help them to widen their team work. This helps the students how to interact each other from ground level. Also this project will become bench marks for other engineering collages and IT collages.

As this project is a simple puzzle solving but by using the concept of image processing we can further improvised our project for solving complex puzzle as well. One of the factors of this project which keeps it in the limelight is the fact that it focuses on the functioning of hardware on the basis of decision made by the software via Bluetooth.

Image processing, is itself an expatiating discipline, and automated system using image processing will add ice to the cake. Adding machine learning aspect can be a vantage. The first robot completes the path receiving the data via Bluetooth. In the same system, another car could be added which completes the same path learning the turns and stops the previous one made. Furthermore, complex puzzles having more than one complete path, can be solved using more advanced algorithms which would also help in reducing the path calculating time.

REFERENCES

J. Brian Burns, Allen R. Hanson, Edward M. Riseman, Extracting Straight Lines, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 4, pp. 425-455, 1986.

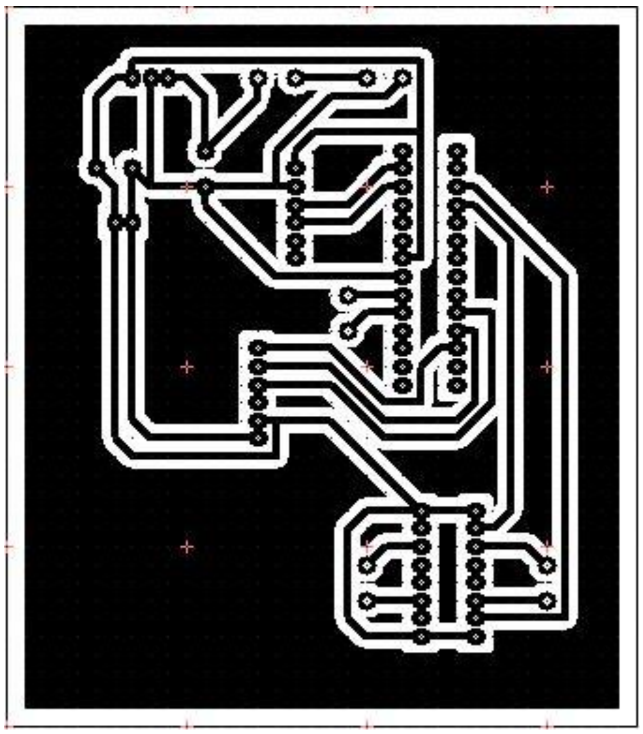
Agn`es Desolneux, Lionel Moisan, Jean-Michel Morel, Meaningful Alignments, International Journal of Computer Vision, vol. 40, no. 1, pp. 7-23, 2000.
http://dx.doi.org/10.1023/A:1026593302236

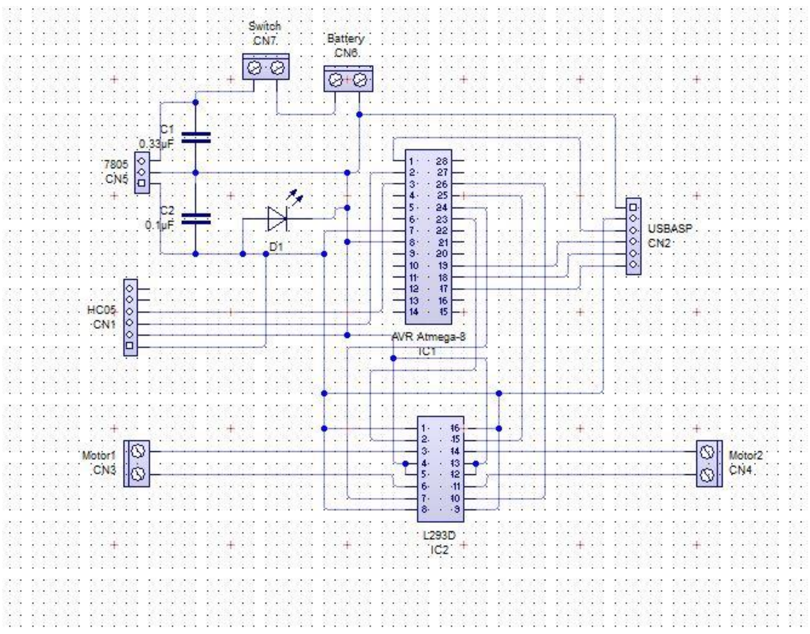
Rafael Grompone von Gioi, Jer´ emie Jakubowicz, Jean-Michel Morel, Gregory Randall ´ , LSD: a Line Segment Detector, Image Processing On Line, 2 (2012), pp. 35–55.
http://dx.doi.org/10.5201/ipol.2012.gjmr-lsd

“The Opencv Tutorials Release 2.4.13.0”,
http://docs.opencv.org/3.0beta/doc/tutorials/tutorials.html, 2016

“Bluetooth Communication between Raspberry Pi and Arduino”, Retrieved August 20, 2016, from *http://www.uugear.com/portfolio/bluetooth-communication-between-raspberry-pi-and-arduino*

APPENDIX A:




Switch
CN7
Battery
CN6
C1
0.33µF
7805
CN5
C2
0.1µF
D1
HC05
CN1
USBASP
CN2
AVR Atmega-8
IC1
Motor1
CN3
Motor2
CN4
L293D
IC2