

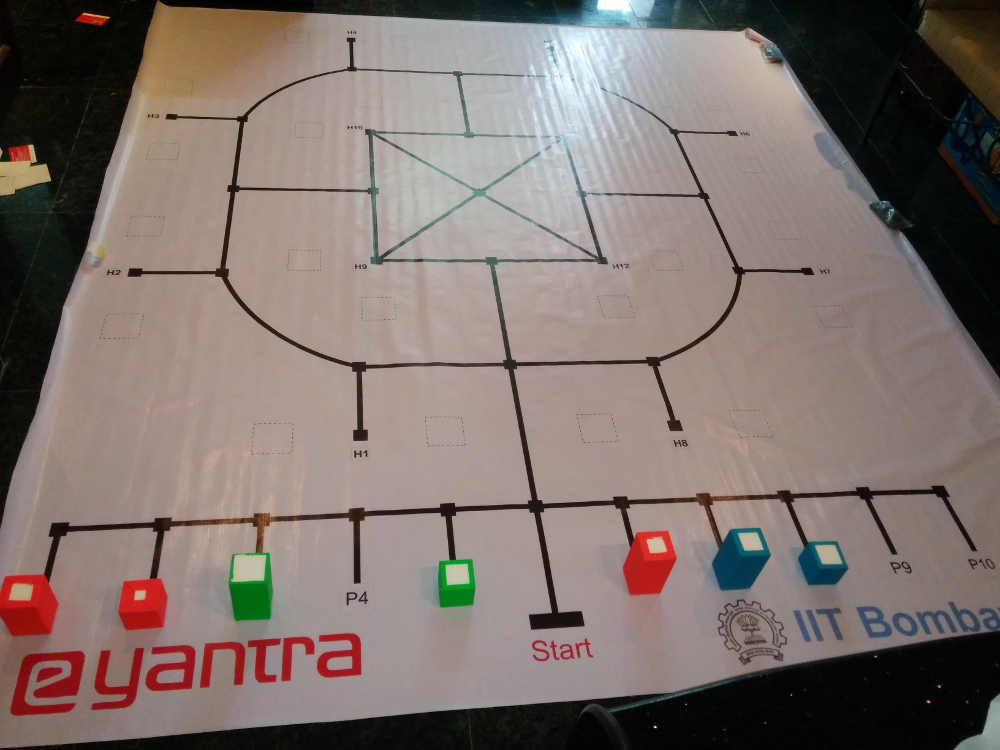
**e-Yantra Robotics Competition - 2015**

**Implementation Analysis – Pizza Delivery Service**

**<Team ID>**

|  |  |
| --- | --- |
| **Team leader name** | Pranjal Aswani |
| **College** | Thakur College of Engineering and Technology |
| **Email** | aswani.pranjal@gmail.com |
| **Date** | 15/1/2016 |

**Preparing the Arena (5)**



**Design Analysis**

**Q-1. What do you understand by “Internal Timer”? and what is the significance of it in this theme. (5)**

An internal timer is a timer controlled by the microcontroller using interrupts in order to separate the execution of time-sensitive code against code that runs and evaluates the functions of the bot that allow it to move, make decisions, and understand its surroundings.

The internal timer is very important because the bot needs to accurately represent how much time it has left for any given order, and the order timeline's solution is optimized using this time. The bot will be scored based on this time too, so it is paramount to ensure that it is reliable.

**Q-2. Make a video in which time displays from 000 to 030 seconds on the Seven Segment Module. (10)**

[All files required to compile task 2's code are available in this zip.](https://drive.google.com/file/d/0B659He5qyintRHhCY00zUDloVVU/view?usp=sharing)

The C code is replicated below. The three files must be linked together and the final hex file is then burned to the atmega2560 processor. This is done using the Makefile(s) seen in the zip.

**SevenSegment.c:**

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include "SevenSegment.h"

void seven\_segment\_pin\_config() {

// Port D upper nibble for CA connections

// Port J for a,b,c,d, e,f,g,DEC as per manual

// Both things above are in decreasing order

// i.e.

// Port J = Pin 7, 6, 5...0

DDRD = DDRD | 0xF0; // set upper nibble of port D to output

DDRJ = DDRJ | 0xFF; // set all bits to output

}

void seven\_port\_init() {

seven\_segment\_pin\_config();

}

void seven\_init\_devices() {

cli(); //Clears the global interrupt

seven\_port\_init(); //Initializes all the ports

sei(); // Enables the global interrupt

}

int seven\_convert\_to\_hex(int num) {

int ret = 0x00;

int values[10];

values[0] = 0x03;

values[1] = 0x9F;

values[2] = 0x25;

values[3] = 0x0D;

values[4] = 0x99;

values[5] = 0x49;

values[6] = 0x41;

values[7] = 0x1F;

values[8] = 0x01;

values[9] = 0x09;

if (num >= 0 && num <= 9) {

ret = values[num];

}

return ret;

}

void seven\_display\_num(int num) {

int digit;

int MIN\_SELECT\_VALUE = 0x10;

int MAX\_SELECT\_VALUE = 0x40;

int select\_value = MIN\_SELECT\_VALUE;

PORTD = PORTD & 0x0F; // Reset upper nibble to 0

PORTD = PORTD | select\_value;

while (select\_value <= MAX\_SELECT\_VALUE) {

digit = num % 10;

PORTJ = seven\_convert\_to\_hex(digit);

\_delay\_ms(5);

select\_value \*= 2;

if (select\_value <= MAX\_SELECT\_VALUE) {

PORTD = PORTD & 0x0F; // Reset upper nibble to 0

PORTD = PORTD | select\_value; // Set upper nibble value to select // apt CA

}

num /= 10;

}

}

**timer.c:**

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include "timer.h"

#include "../buzzer/buzzer.h"

// TIMER4 initialize - prescale:1024

// WGM: 0) Normal, TOP=0xFFFF

// desired value: 1Hz

// actual value: 1.000Hz (0.0%)

void timer\_timer4\_init() {

TCCR4B = 0x00; // stop

TCNT4H = 0x1F; // Counter higher 8 bit value

TCNT4L = 0x01; // Counter lower 8 bit value

OCR4AH = 0x00; // Output Compair Register (OCR)- Not used

OCR4AL = 0x00; // Output Compair Register (OCR)- Not used

OCR4BH = 0x00; // Output Compair Register (OCR)- Not used

OCR4BL = 0x00; // Output Compair Register (OCR)- Not used

OCR4CH = 0x00; // Output Compair Register (OCR)- Not used

OCR4CL = 0x00; // Output Compair Register (OCR)- Not used

ICR4H = 0x00; // Input Capture Register (ICR)- Not used

ICR4L = 0x00; // Input Capture Register (ICR)- Not used

TCCR4A = 0x00;

TCCR4C = 0x00;

TCCR4B = 0x04; // start Timer

}

// This ISR can be used to schedule events like refreshing ADC data, LCD data

ISR(TIMER4\_OVF\_vect) {

// TIMER4 has overflowed

TCNT4H = 0x1F; // reload counter high value

TCNT4L = 0x01; // reload counter low value

timer\_1s\_magic();

}

void timer\_init\_devices() {

cli(); // Clears the global interrupts

timer\_timer4\_init();

TIMSK4 = 0x01; // timer4 overflow interrupt enable

sei(); // Enables the global interrupts

}

**task2.c:**

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include "../timer/timer.h"

#include "../SevenSegment/SevenSegment.h"

int count = 0;

void timer\_1s\_magic() {

count++;

if (count >= 30) {

count = 30;

}

}

int main() {

timer\_init\_devices();

seven\_init\_devices();

while(1) {

seven\_display\_num(count);

\_delay\_ms(1);

}

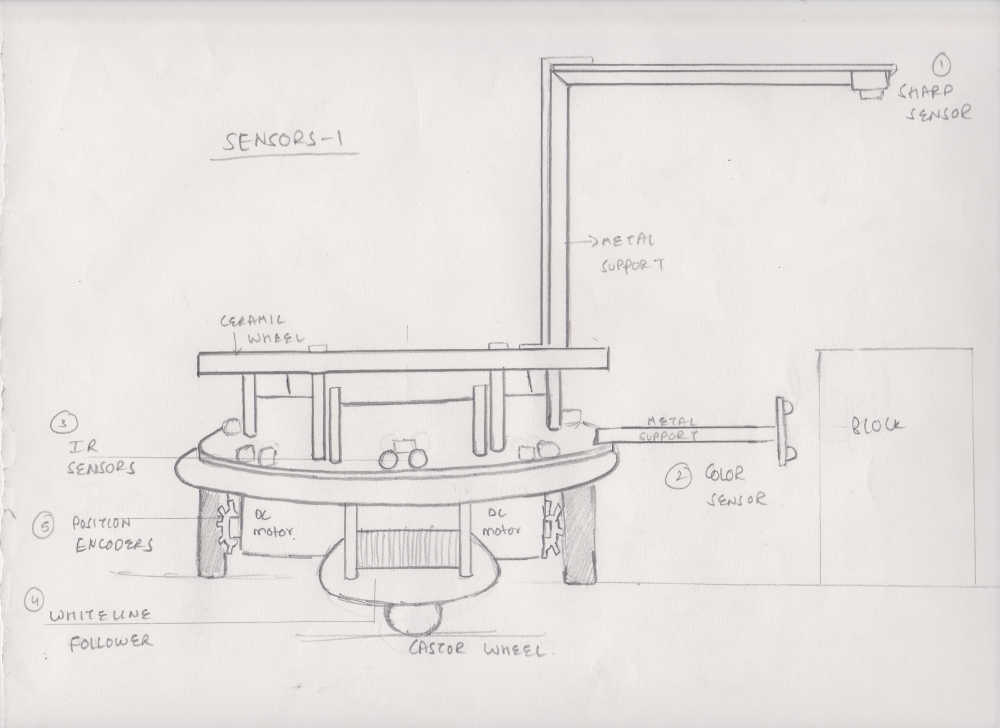
}

**Q-3. Draw a labeled diagram to explain how you have planned to place the sensors on/around the robot? (5)**

We have planned to place the **sharp sensor on the left side** of the bot, 21cm above the ceramic plate of the bot, and 11cm away from the edge of the left side of the bot as shown in the diagram.

The **colour sensor will be placed on the left side** approximately 8cm away from the edge of the left side of the bot.

All the other sensors will be mounted in their original positions.



**Q-4. Teams have to make the robotic arm for picking up/placing the packages in the arena.**

1. **Choose an option you would like to use to position the robotic arm on the robot and why? (2)**

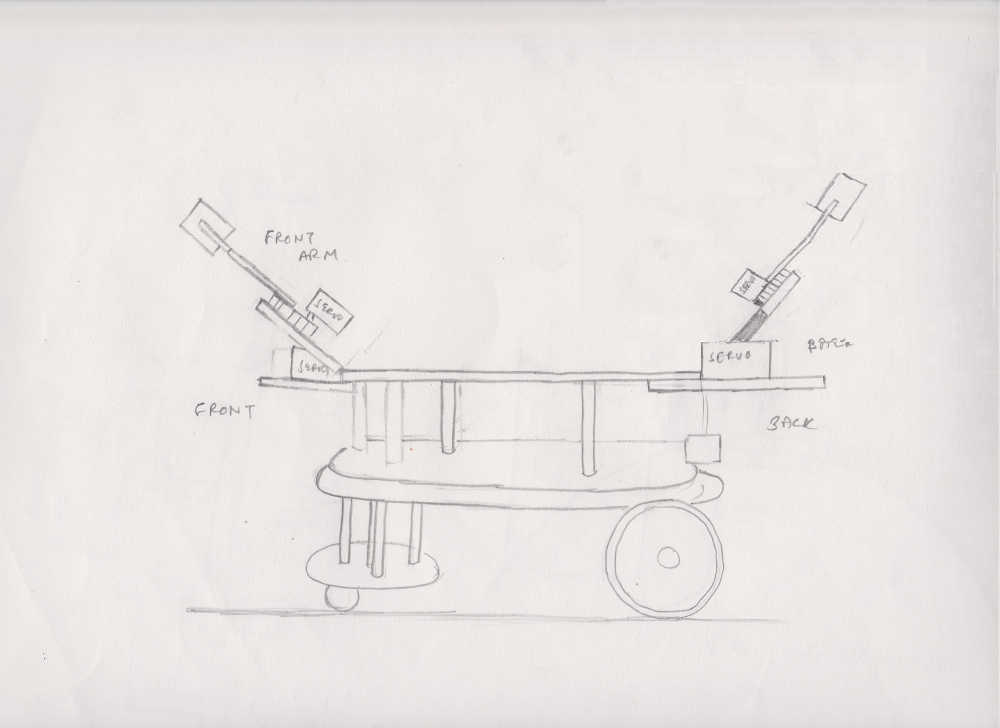
1. **Front 2. Back 3. Right/Left 4. On both sides**

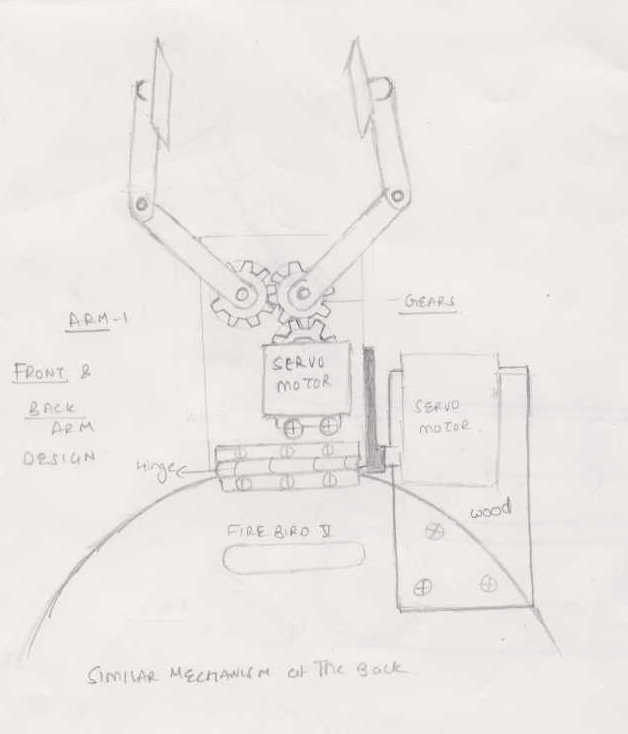
We chose to mount the robotic arm on the **front** of the robot because of the following reasons:

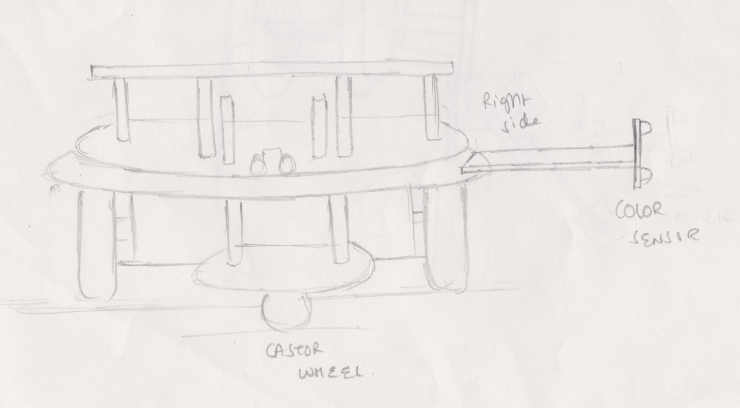
* The bot cannot traverse sideways directly; it would need to stop in the correct position, and rotate 90 degrees if the arm was placed on either side. This would add wasteful time to the bot's deliveries in a time-sensitive competition.
* If we mount the arm on either sides, we risk toppling over other pizza boxes at the pizza store because of restrictions in space.
* We have mounted our sharp sensor on the side, which also restricts the space we have to mount the arm on the side.
* If we mounted 2 arms, on both sides of the bot, it would not be able to traverse sideways directly, and might risk toppling over pizza boxes.
* Having 2 arms, one on the front, and one on the back allows us to carry 2 pizza boxes at once if required, and leaves us with little to no disadvantage if it isn't required. Power consumption should remain low, as the arms will be turned off when they aren't required.

Seeing these reasons, we've decided to mount an arm on both the **front** and **back** as we think that the possible advantages overcome the disadvantages.

**b. Draw a diagram to show the robotic arm and how it is mounted on the robot. Also show the mounting of the color sensor. (5)**







**Q-5. Choose the actuator you will use to design the robotic arm. (3)**

1. **DC-Motor 2. Servo Motor 3. Stepper Motor 4. Others**

**Answer:** 2. Servo Motor

We have chosen the servo motor because they are the most accurate and reliable motors for the specific purpose of building an arm. A DC motor is an open-loop system, which means we'd require specific sensors (like position encoders) to track how far the motor had moved. A servo motor on the other hand, is a closed-loop system, and allows us to move the motor with the kind of precision required to control a robotic arm. We don't use stepper motors because stepper motors are generally very expensive, and they generally provide a lot more precision and torque than we require for the arm.

**Q-6**

**a. What is the principle of operation of the color sensor? (5)**

The given programmable color light-to-frequency converter is a combination of silicon photo-diodes (where a current is induced due to light) and a current-to-frequency converter on a single integrated circuit. The given light-to-frequency converter takes inputs from photo-diodes arranged in shape of a matrix. The given color sensor also has four LEDs at each of its corners which illuminate when the color sensor is turned on. The light emitted from the LEDs is reflected back from the surface of the pizza blocks and is taken up as the input of the photo-diodes. Pins S2 and S3 are used to select which group of photo-diodes are active (S3=L, S2=L for red, S3=H, S2=L for blue, S3=H, S2=H for green, and S3=L, S2=H for clear). The light is then converted into an analog frequency reading and the pins S0 and S1 are used to control the scaling factor of the device. The color is detected by the intensity of the light detected by the array of photo-diodes and the output is a square wave with the frequency directly proportional to light intensity.

**b. Explain frequency scaling. Why is it necessary? (5)**

The color sensor's output is a PWM wave where the width is directly proportional to the intensity of the light of the frequency selected using pins S3 and S2. Every wave has a frequency, i.e. the number of oscillations per second. The PWM wave also has a frequency, which is scaled down to 20% of its original frequency, using internal frequency dividers. We select this scale using pins S1 and S0. Our external input, which counts the pulses received by this PWM wave runs at a frequency below the PWM wave's full-scale frequency. Because of this, we might skip recording pulses because we aren't sampling the wave often enough. To avoid this, we scale the PWM wave's frequency down to 20% so that our interrupt can sample the wave satisfactorily, and we receive an accurate pulse reading.

**c. How will you identify Red, Blue and Green colors from the values you get from the color sensor? Explain your algorithm to identify the three colors (Red, Blue and Green). (5)**

Step 1: Setup Port H as output

Step 2: Setup Port D, Pin 0 as an interrupt

Step 3: Setup interrupt to increment a “pulse” variable

Step 4: Create a function that returns the character 'r', 'g' or 'b' for red, green or blue respectively

Step 5: Set the “pulse” variable to 0

Step 6: Set pins S3=0 and S2=0 to select for “red” pulses

Step 7: Store output in a variable called “red” after 100ms

Step 8: Repeat this process for green and blue as well, with their respective S3 and S2 combinations

Step 9: Select the highest pulses received and return the apt character ('r' for “red”, etc.)

**Actual code:**

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include "color\_sensor.h"

volatile unsigned long int color\_pulse = 0;

const int color\_data\_time = 100;

void color\_pin\_config() {

DDRH = DDRH | 0xF0; // Set upper nibble of Port H as output

DDRD = DDRD & 0xFE; // Set pin 0 of Port D as input (interrupt)

}

void color\_interrupt\_init() {

// Enable INT0 (at D0)

cli();

EICRA = EICRA | 0x02; // INT0 is set to trigger with falling edge

EIMSK = EIMSK | 0x01; // Enable Interrupt INT0 for color sensor

sei();

}

ISR(INT0\_vect) {

color\_pulse++;

}

int color\_collect\_pulses() {

color\_pulse = 0;

\_delay\_ms(color\_data\_time);

return color\_pulse;

}

int color\_get\_red() {

PORTH = PORTH & 0xBF; //set S2 low

PORTH = PORTH & 0x7F; //set S3 low

return color\_collect\_pulses();

}

int color\_get\_green() {

PORTH = PORTH | 0x40; //set S2 High

PORTH = PORTH | 0x80; //set S3 High

return color\_collect\_pulses();

}

int color\_get\_blue() {

PORTH = PORTH & 0xBF; //set S2 low

PORTH = PORTH | 0x80; //set S3 High

return color\_collect\_pulses();

}

void color\_sensor\_scaling() {

PORTH = PORTH | 0x10; //set S0 high

PORTH = PORTH | 0x20; //set S1 high

}

void color\_init\_devices() {

color\_pin\_config();

color\_interrupt\_init();

color\_sensor\_scaling();

}

char color\_get() {

// It will return the following characters:

// u for unknown, by default (if something went wrong)

// r for red

// g for green

// b for blue

// It'll ideally take 300ms for this function to run, since we collect 100ms of data for each filter

char ret = 'u';

int red, green, blue;

red = color\_get\_red();

green = color\_get\_green();

blue = color\_get\_blue();

if (red > green && red > blue) {

ret = 'r';

}

else if (green > red && green > blue) {

ret = 'g';

}

else if (blue > green && blue > red) {

ret = 'b';

}

return ret;

};

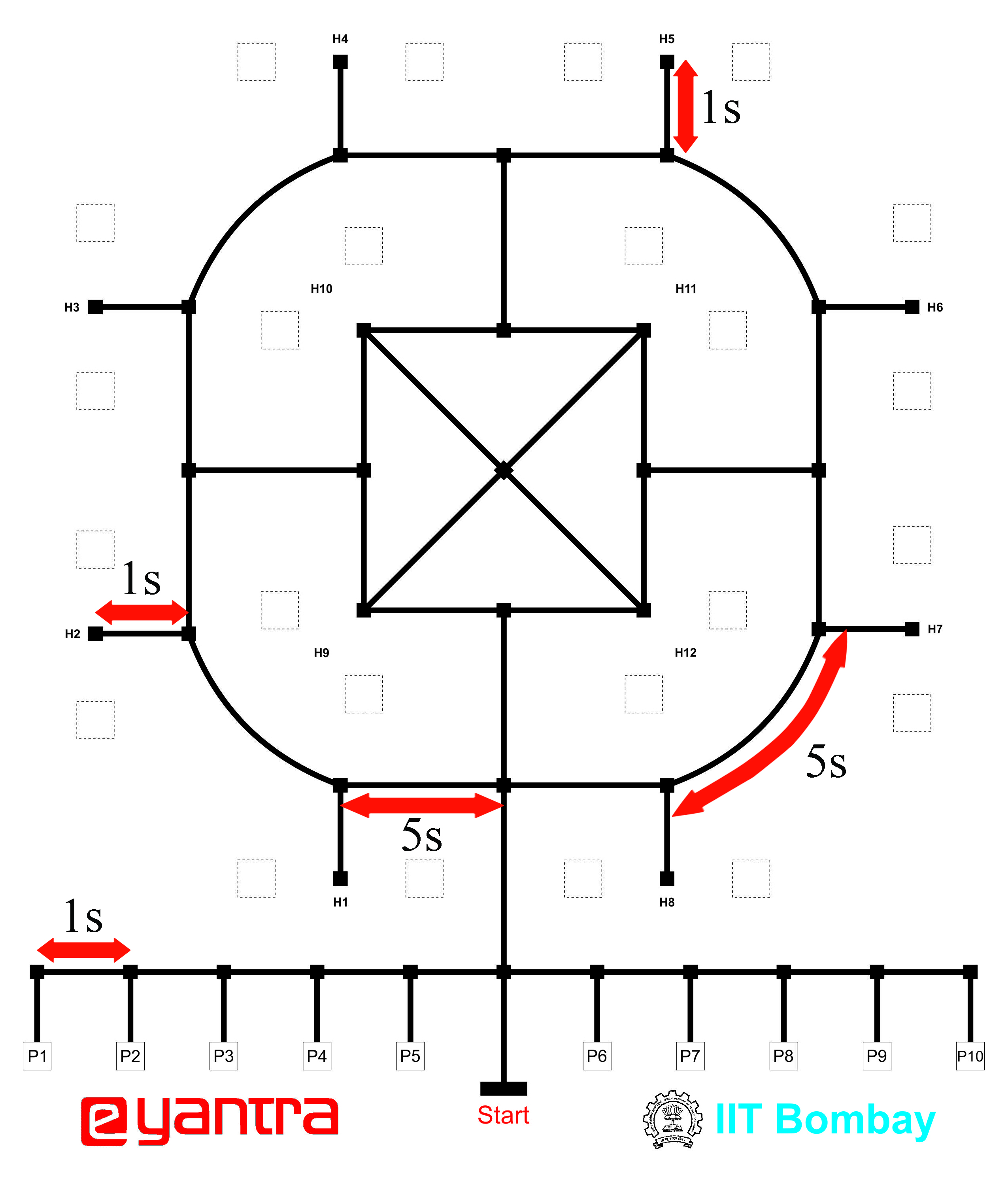
**Note: The timings mentioned below are purely hypothetical. Your robot timings may be shorter or longer than given below.**

**Consider the following assumptions:**

* **The time taken by the robot to travel between the Nodes of Pizza Shop in arena is 1 second.**
* **The time taken by the robot to travel between the Nodes above Pizza Shop in arena is 5 seconds.**
* **The time taken by the robot to travel from Node before the Front Door to Front Door is 1 second.**

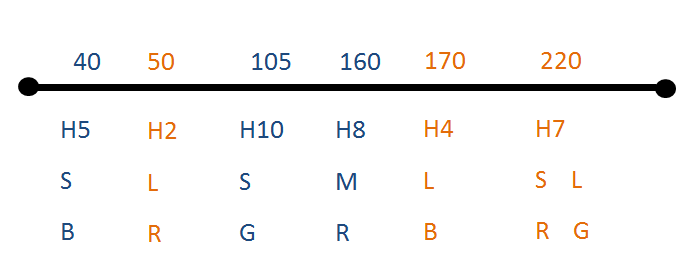
**Refer to Figure 1.**

**Note: Time for picking and placing the Pizza is not considered in the traversal time.**



**Figure 1: Time for Traversal between Nodes**

**Q-7 Given an Order Timeline as shown in Figure 2 how you will deliver each and every Pizza such that you score the maximum points as per the Formula in “Scoring System” in the rulebook. (20)**

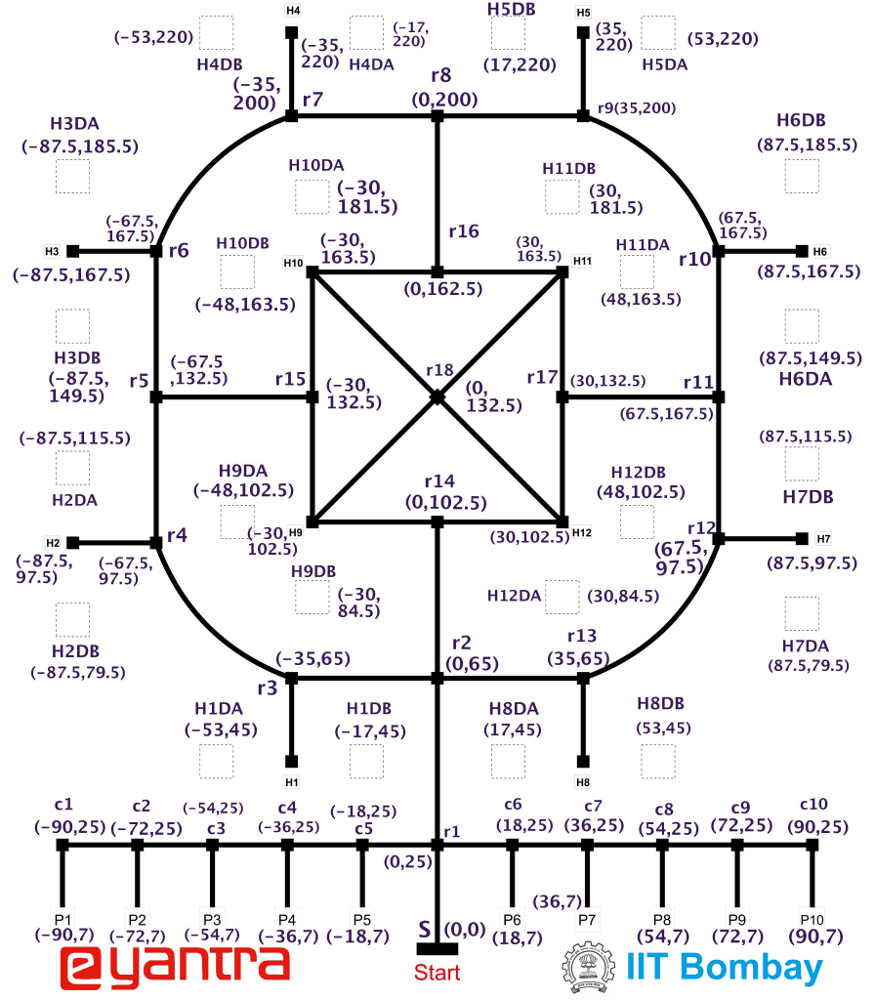
****

**Figure 2: Order Timeline**

**The placement of Pizzas at Pizza Shop is given below:**

**Small Blue Pizza that is to be delivered at H5 is placed at P8 (S, B is at P8).**

**Similarly: L, R is at P6 S, G is at P5 M, R is at P1 L, B is at P7 S, R is at P2 L, G is at P3**



**Answer:** Using the image below for reference, we show the path the bot will take and the time it takes.

We assume that all deliveries are correct in terms of the bot selecting the correct pizza, delivering it to the correct house, and delivering it within the deposition zone.

We also assume the following variables for certain actions. Different assumptions would change the maximum score achieved.

The variables are:

T = Time taken for robot to turn 90 degrees = 1s

A = Time taken for robot to lower / raise arm = 1s

C = Time taken for robot to close / open grip = 0.5s

Our solution to the timeline is as follows:

**Deliver Large Red pizza to H2**

Step 1) r1->c6

Time taken: 1s

Step 2) TACAT

Time taken: 4.5s

Step 3) c6->r1

Time taken: 1s

Step 4) r1->r2->r3->r4->h2

Time taken: 15s + 1s = 16s

Step 5) TAC

Time taken: 2.5s

Total time taken for operation = 25s

Cumulative tips received = 1 (for delivery in time)

**Go back to pizza counter:**

Step 1) AT

Time taken: 2s

Step 2) h2->r1

Time taken: 16s

Total time for operation = 18s

Cumulative time taken = 43s

**Deliver Small Blue pizza to H5:**

Step 1) r1->c8

Time taken: 3s

Step 2) TACAT

Time taken: 4.5s

Step 3) c8->r1

Time taken: 3s

Step 4) r1->r2->r13->r12->r11->r10->r9->h5

Time taken: 30s + 1s = 31s

Step 5) TAC

Time taken: 2.5s

Total time taken for operation = 44s

Cumulative time taken = 87s

Cumulative tips received = 2

**Go back to pizza counter:**

Step 1) AT

Time taken: 2s

Step 2) h5->r1

Time taken: 31s

Total time taken for operation = 33s

Cumulative time taken = 120s

**Deliver Small Green pizza to H10**

Step 1) r1->c5

Time taken: 1s

Step 2) TACAT

Time taken: 4.5s

Step 3) c5->r1

Time taken: 1s

Step 4) r1->r2->r14->h9->r15->h10

Time taken: 25s

Step 5) TAC

Time taken: 2.5s

Total time taken for operation = 34s

Cumulative time taken = 154s

Cumulative tips received = 3

**Go back to pizza counter:**

Step 1) AT

Time taken: 2s

Step 2) h10->r1

Time taken: 25s

Total time taken for operation = 27s

Cumulative time taken = 181s

**Deliver Medium Red pizza to H8**

Step 1) r1->c1

Time taken: 5s

Step 2) TACAT

Time taken: 4.5s

Step 3) c1->r1

Time taken: 5s

Step 4) r1->r2-r13->h8

Time taken: 11s

Step 5) TAC

Time taken: 2.5s

Total time taken for operation = 28s

Cumulative time taken = 209s

Cumulative tips received = 4

**Go back to pizza counter:**

Step 1) AT

Time taken: 2s

Step 2) h8->r1

Time taken: 11s

Total time taken for operation = 13s

Cumulative time taken = 222s

**Deliver Small Red and Large Green pizza to H7**

Step 1) r1->c2

Time taken: 4s

Step 2) TACAT (front arm)

Time taken: 4.5s

Step 3) c2->c3

Time taken: 1s

Step 4) TACAT (back arm)

Time taken: 4.5s

Step 5) c3->r1

Time taken: 3s

Step 6) r1->r2->r13->r12->h7

Time taken: 15s + 1s = 16s

Step 7) TAC (simultaneous delivery)

Time taken: 2.5s

Total time taken for operation = 35.5s

Cumulative time taken = 257.5

Cumulative tips received = 6 (incremented by 2 for 2 pizza deliveries)

**Go back to pizza counter:**

Step 1) AT

Time taken: 2s

Step 2) h7->r1

Time taken: 16s

Total time taken for operation = 18s

Cumulative time taken = 275.5s

**Deliver Large Blue pizza to H4:**

Step 1) r1->c7

Time taken: 2s

Step 2) TACAT

Time taken: 4.5s

Step 3) c7->r1

Time taken: 2s

Step 4) r1->r2->r3->r4->r5->r6->r7->h4

Time taken: 31s

Step 5) TAC

Time taken: 2.5s

Total time taken for operation = 42s

Cumulative time taken = 317.5

Cumulative tips received = 6 (no tip for late delivery)

**Formula** = (600 - T) + (CPD \* 50) + (CPCD \* 50) + (TIP \* 50) + (CD \* 10) - (IPD \* 20) - (P\*50)

**According to our solution, the variables are**:

T = 317.5

CPD = 7

CPCD = 7

TIP = 6

CD = 7

IPD = 0

P = 0

Therefore:

**Score = (600 - 317.5) + (7 \* 50) + (7 \* 50) + (6 \* 50) + (7 \* 10)**

**Score = 1352.5**

**Algorithm Analysis**

**Q-1 Draw a flowchart illustrating the major functions that are used. (20)**

**pos\_encoder():** Increments a variable to count the distance traveled.

**init\_pwm():** Initializes the PWM wave required for the motor control.

**rotate\_towards\_node(source\_node, target\_node):** Calculate the shortest angle between the source\_node and target\_node and rotates the bot toward it within a certain angle of tolerance, relying on the white line sensors to align with the black line.

**rgb\_led\_glow(color):** Make the RGB led glow red, green, or blue according to the characters passed: 'r', 'g', 'b'.

**setup\_interrupts():** Initialize interrupt service routines to run every 1 second, and call the functions **timer\_run()** and **analyze\_timeline()**.

**timer\_run():** Increments the timer counter.

**analyze\_timeline():** Determine the order of delivery of pizzas every 1 second, based on any new information that is found. This allows the bot to make decisions dynamically. For example, when the bot doesn't know where a pizza is, it will assume a certain random or average time to find the pizza. But if the bot finds the pizza faster than expected, it might prioritize its delivery, since it may maximize the score to do so.

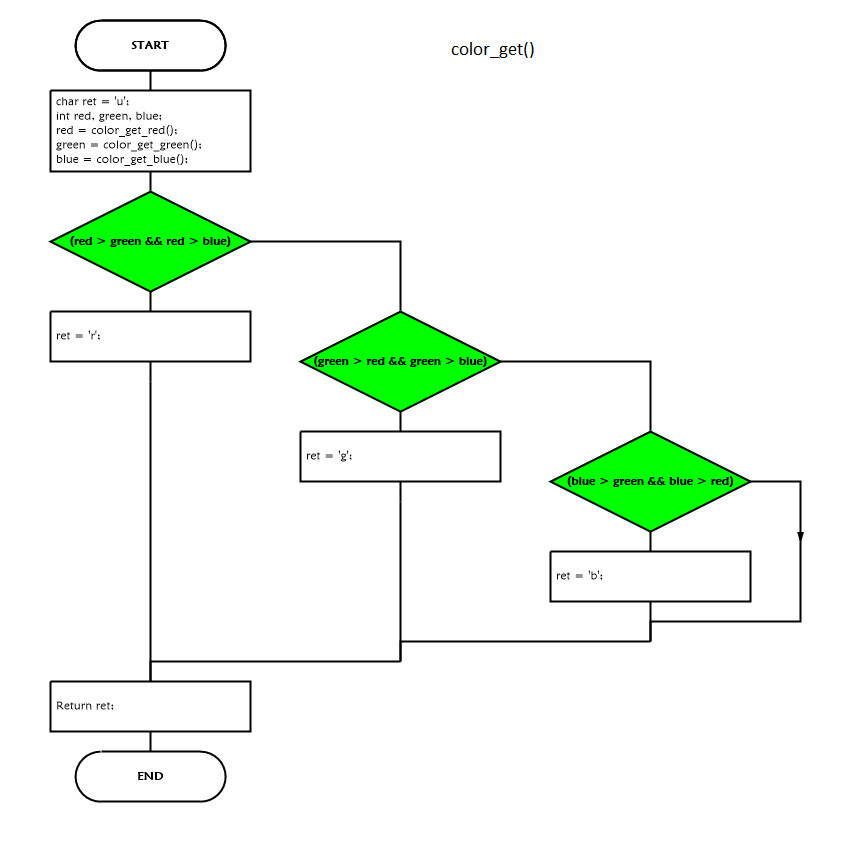
**find\_path():** Use Dijkstra's algorithm to find the shortest route from one node to another.

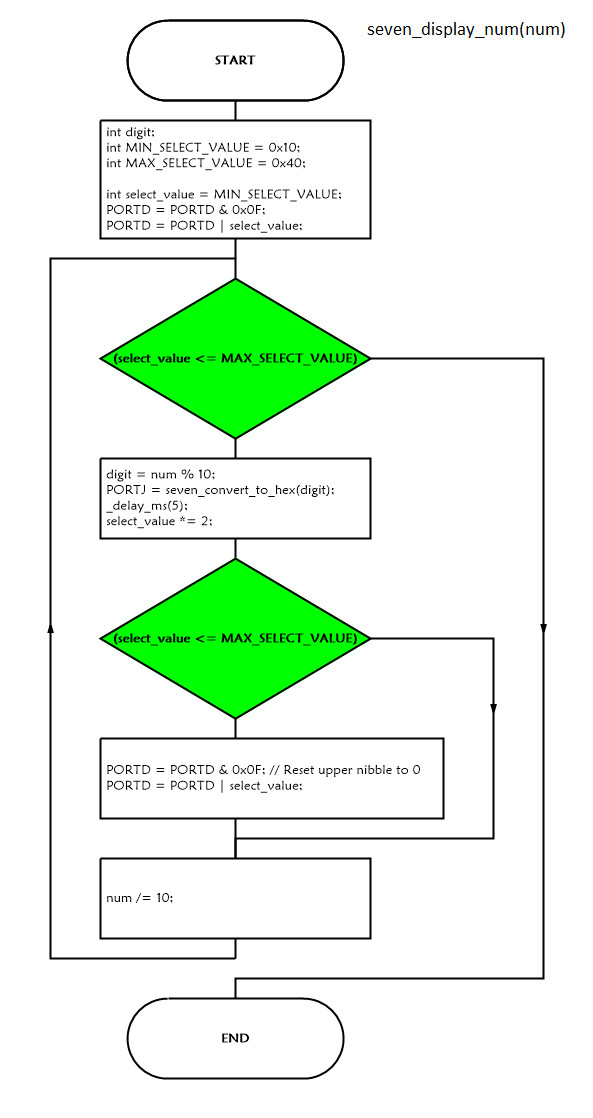
**white\_line\_align():** Keeps the bot aligned with the black line, using the white line sensors.

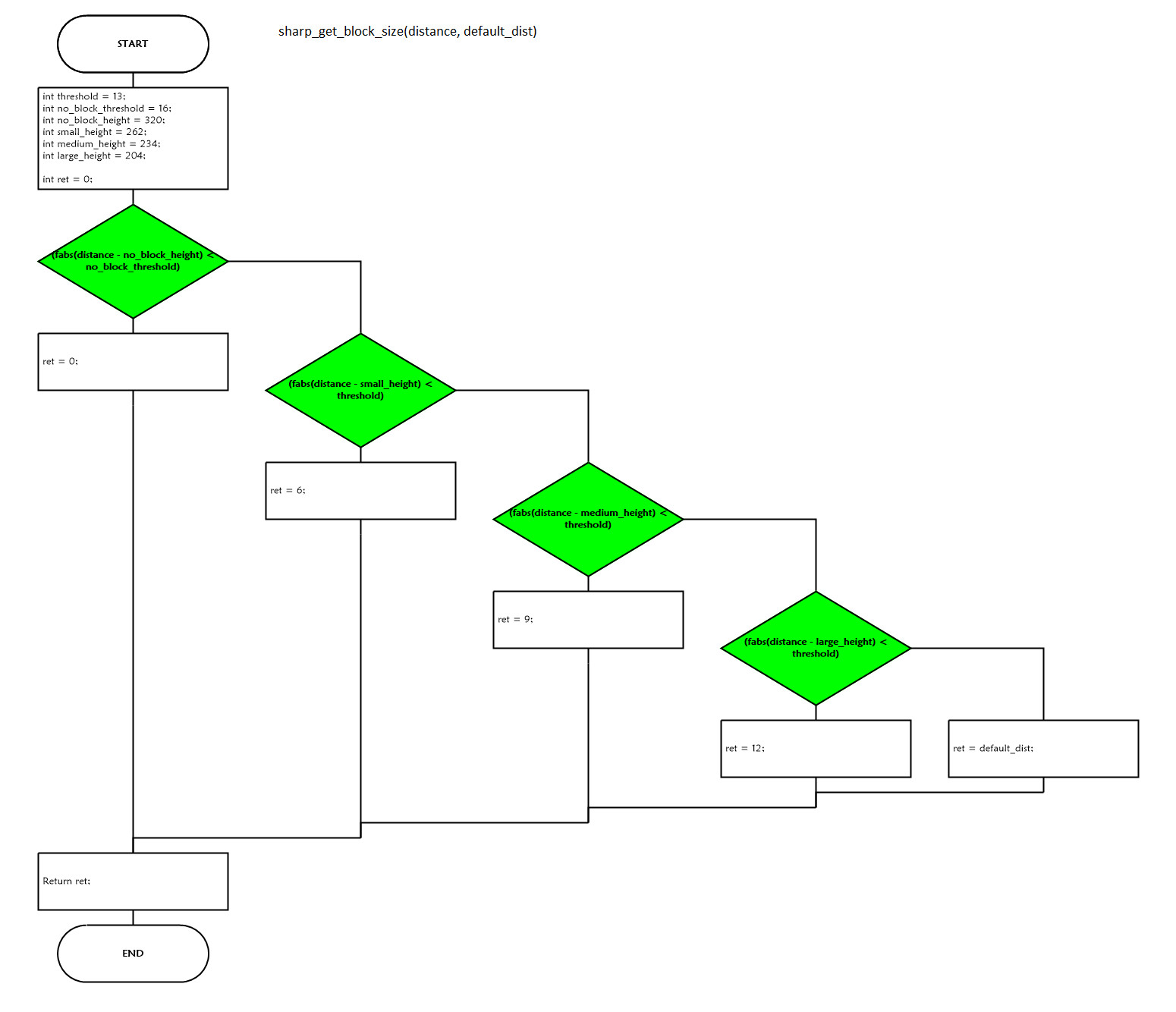
**motor\_forward():** Moves both motors at the same speed in order to move the bot forward.

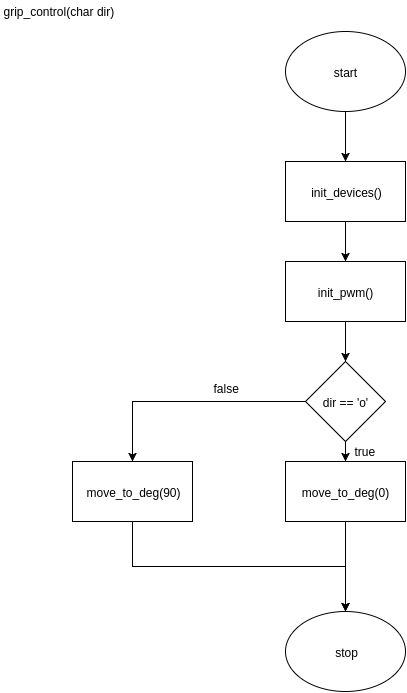
**move\_curve(source\_node, target\_node):** Uses hard-coded values combined with the white line sensor to follow the curves in the map.

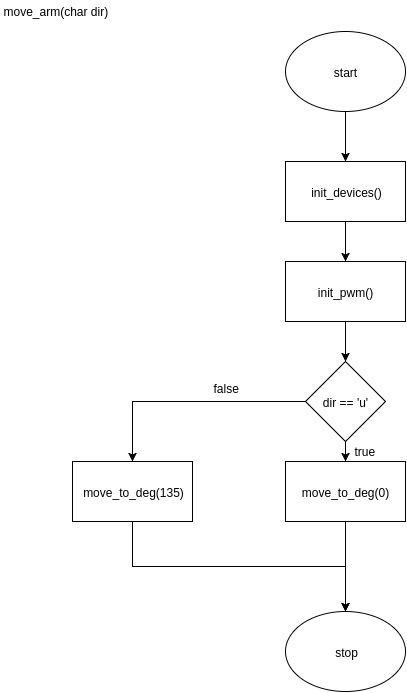
**seven\_convert\_to\_hex():** Converts a numerical value to the hex value required by the pins on the seven segment display.

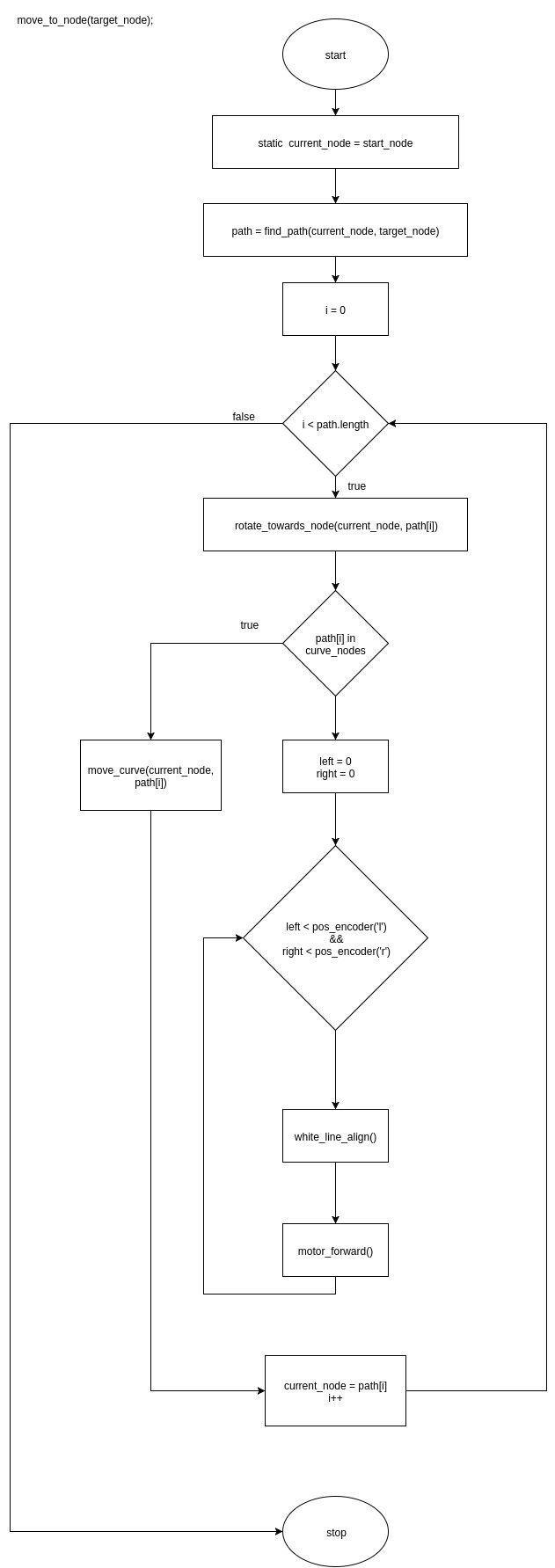
****

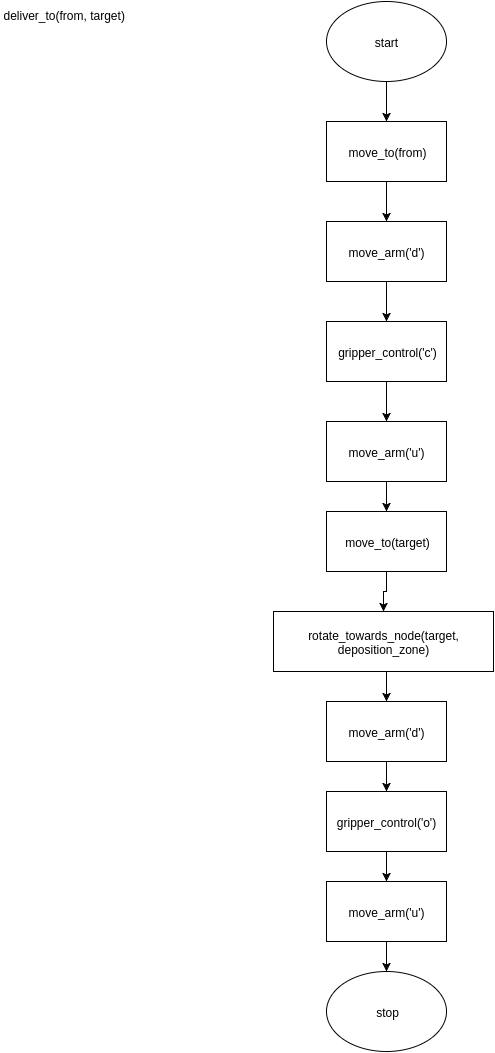
****

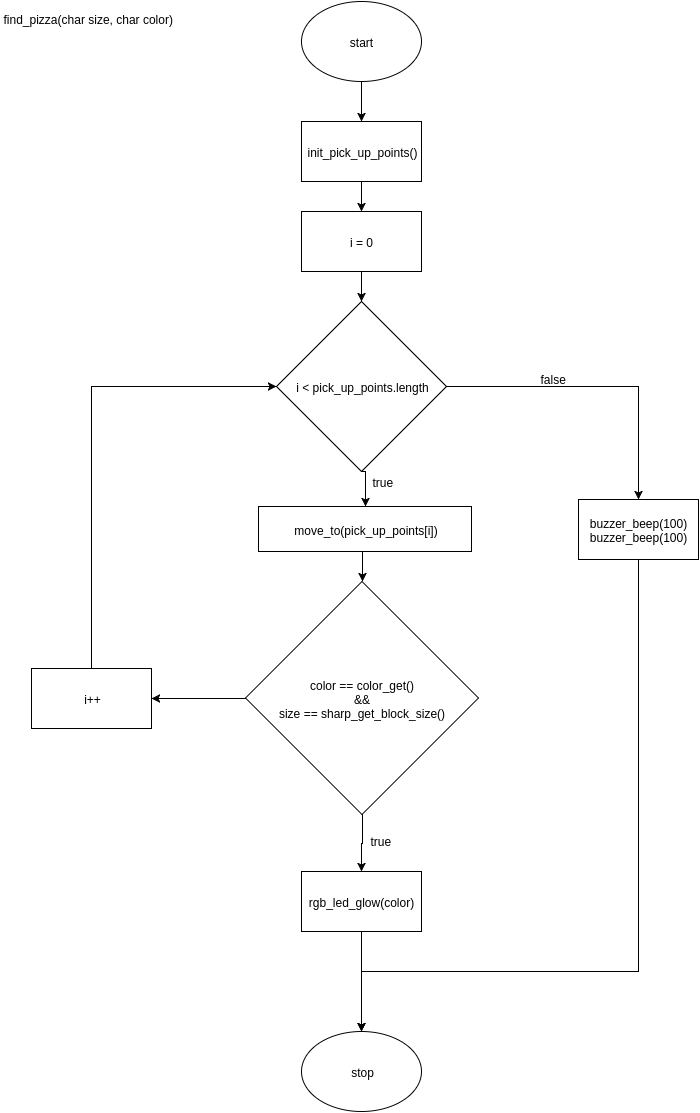
****

****

****

****

****



**Q-2 Draw a flowchart illustrating main function of your code. (10)**

**init\_devices()**: Initialize all the ports required by every module as output or input pins, as appropriate.

**setup\_interrupts()**: Function which initializes interrupts to run every 1 second. The interrupts call a function to run the background timer (which counts how much time has elapsed) and a function to analyze the order timeline given the details we have currently.

**int pizzas\_remaining**: initialize global variable

**read\_timeline()**: Read the timeline in, and update appropriate variables like pizzas\_remaining.

**find\_pizza(current\_pizza)**: Function which finds a pizza by moving the bot to the pizza pick-up points and uses the sharp sensor and colour sensor to identify pizzas. Upon finding the correct pizza, this function will also glow the appropriate color on the RGB LED, and save the pizza's location internally. (current\_pizza is a global variable which is set in the “analyze\_timeline()” function which runs in an interrupt.

If the pizza wasn't found, it beeps the buzzer twice.

**deliver\_to(current\_target)**: This function will pick up the pizza using either the front or back arm, move to the delivery spot, and drop the pizza in the deposition zone. Also, freeze the timer for 5s.

**buzzer\_beep(int X)**: Functions that turns the buzzer on for X ms and off for X ms. This lets us beep the buzzer for 5s at the end.

**stop\_timer()**: Freeze the timer display indefinitely.

