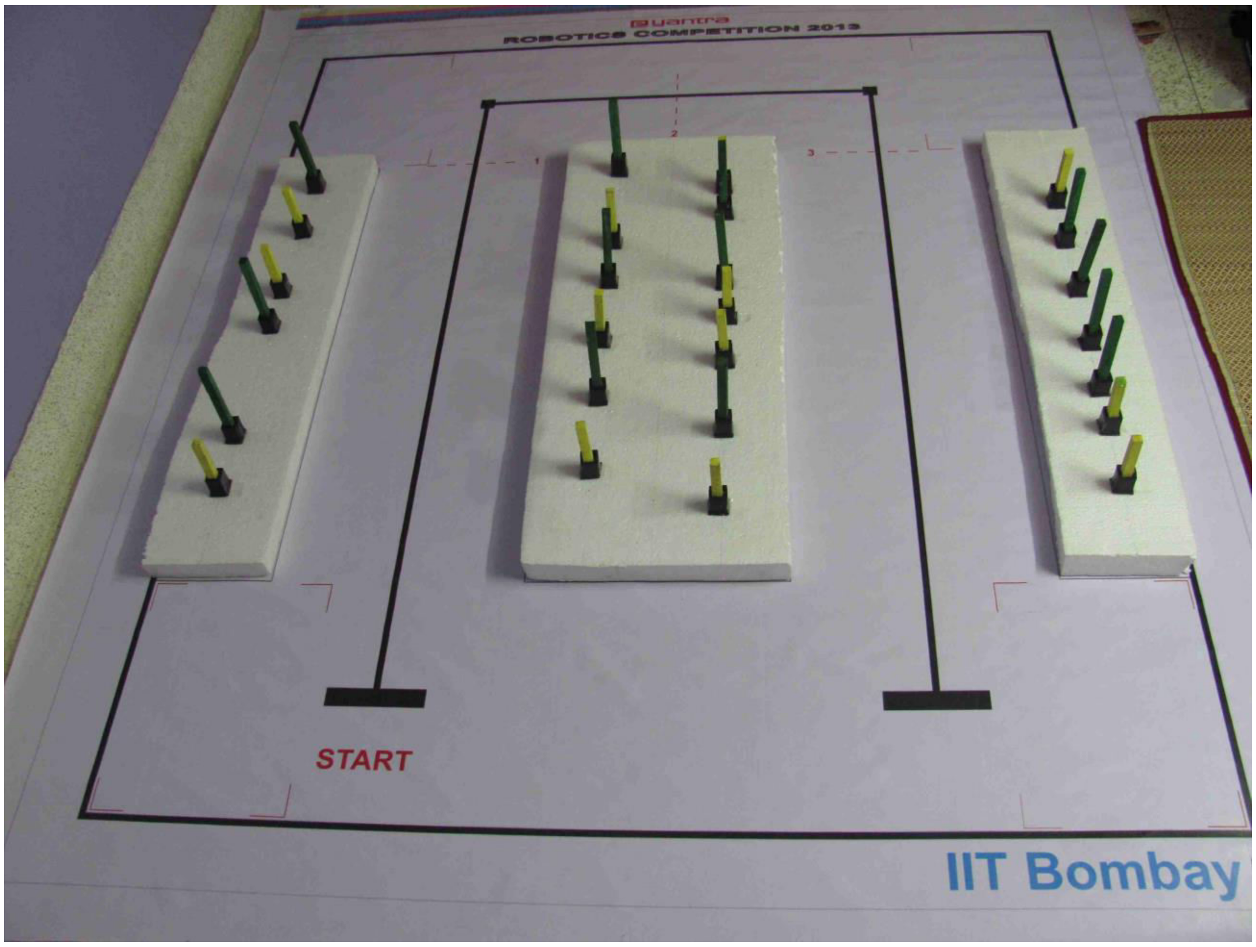
e-Yantra Robotics Competition - 2013

Implementation Analysis- Weeding Robot

e-YRC#211-WD

Team leader name	SOUMYADIP GHOSH
College	JADAVPUR UNIVERSITY, KOLKATA
Email	soumyadip93@gmail.com
Date	27/01/2014

Preparing the Arena (10)



Design Analysis

Q-1 How will you differentiate between the weeds and plants using the sensors available to you in the kit? (10)

In this theme, our goal is to collect the weeds from the green house. In order to do this, the main challenge isto differentiate the weeds from the plants. Comparing the characteristics of weeds and plants, the unique feature to differentiate them without any ambiguity is their height. The weed height is 10 cm and plant height is 15 cm. So there is a clear 5 cm height difference between the weed and plant. Now we had 8 IR proximity sensors and 2 sharp sensors and our next move was on how to identify the height difference with these sensors.But a lot of challenges appeared as follows:-

1. The IR proximity sensors mounted on the robot PCB can't be used directly as it is below the trough platform where plants and weeds are placed.
2. If we don't use IR proximity sensor, we have only 2 sharp sensors for our theme. According to our plan, to identify the weed on the basis of height, 2 sensors are to be placed - one below 10 cm and the other between 10 cm and 15 cm. So using only 2 sharp sensors, we can only detect weed at one side of the black line. However, for simultaneous detection along the both side of the blackline, only 2 sharp sensors are not sufficient.
3. If we use IR proximity sensor, the range of this type of sensor is less than 10cm. But the distance from the black line to weed is 30 cm,so these sensors can't detect anything if we mount it on the robot body.

To overcome the above problems, we use both the sharp and proximity sensors.So we have to de-solder 4 proximity sensors from the delicate robot PCBwith utmost care to serve our purpose .The sharp sensors are placed at a height below 10 cm and the IR proximity sensors are placed between 10 cm and 15 cm. The robot is supposed to follow the black line and move in the forward direction. If the sharp sensor value crosses a threshold (a pre-set value for a specific distance), we will infer that it detects a weed or a plant. Now 2 IRproximity sensors,placed on each side of the robot,will sense whether it is a weed or a plant. If there is a plant, the IR proximity sensor value will cross the selected threshold due to reflection from plant. However, if the value stays below the threshold, no reflection occurs between the 10 and 15 cm height, so the obstacle must be a weed. Hence the combined reading of bothtypes of sensors will determine whether it is a weed or plant. As the range of IR proximity sensor is small, 2 sensors on each side are used for better reliability.

Q-2 Draw a labeled diagram to explain how you have planned to place the sensors on/around the robot? (15)

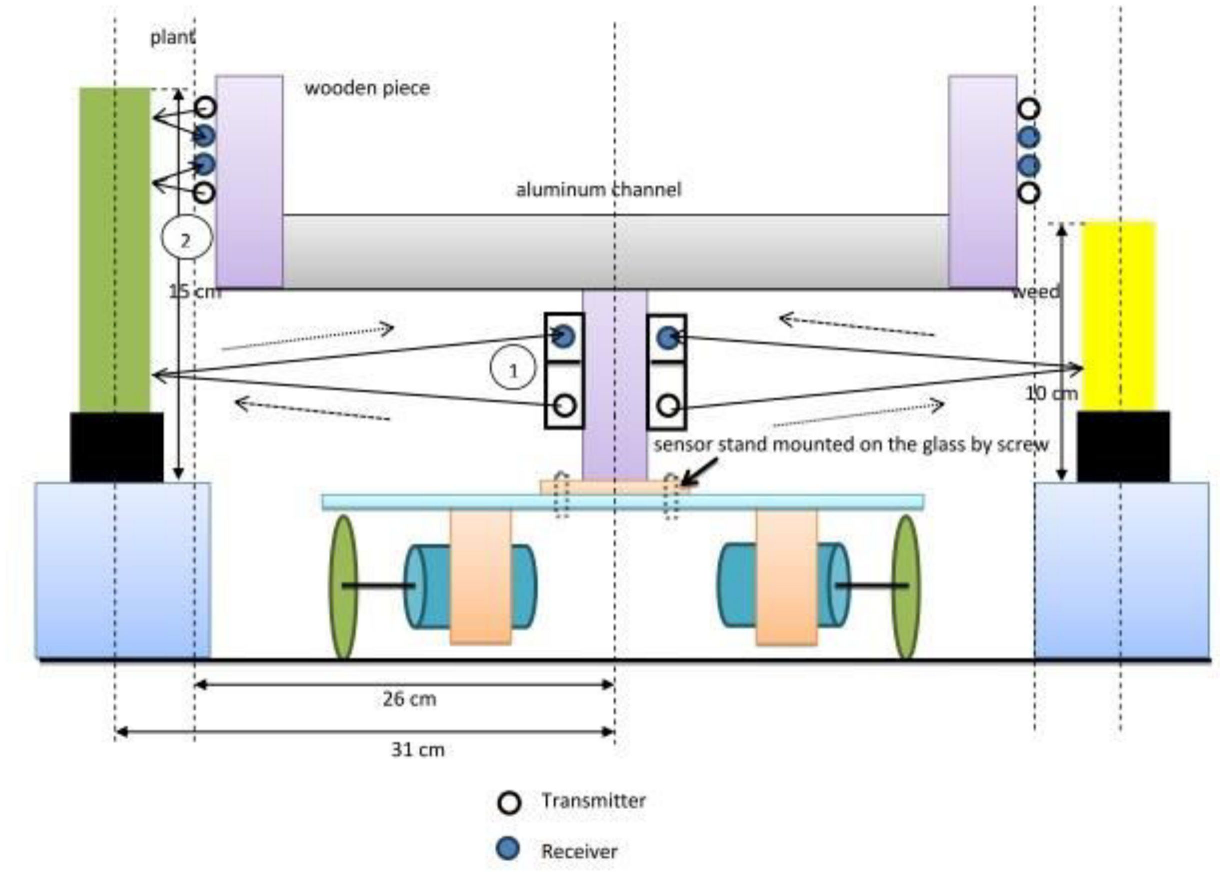
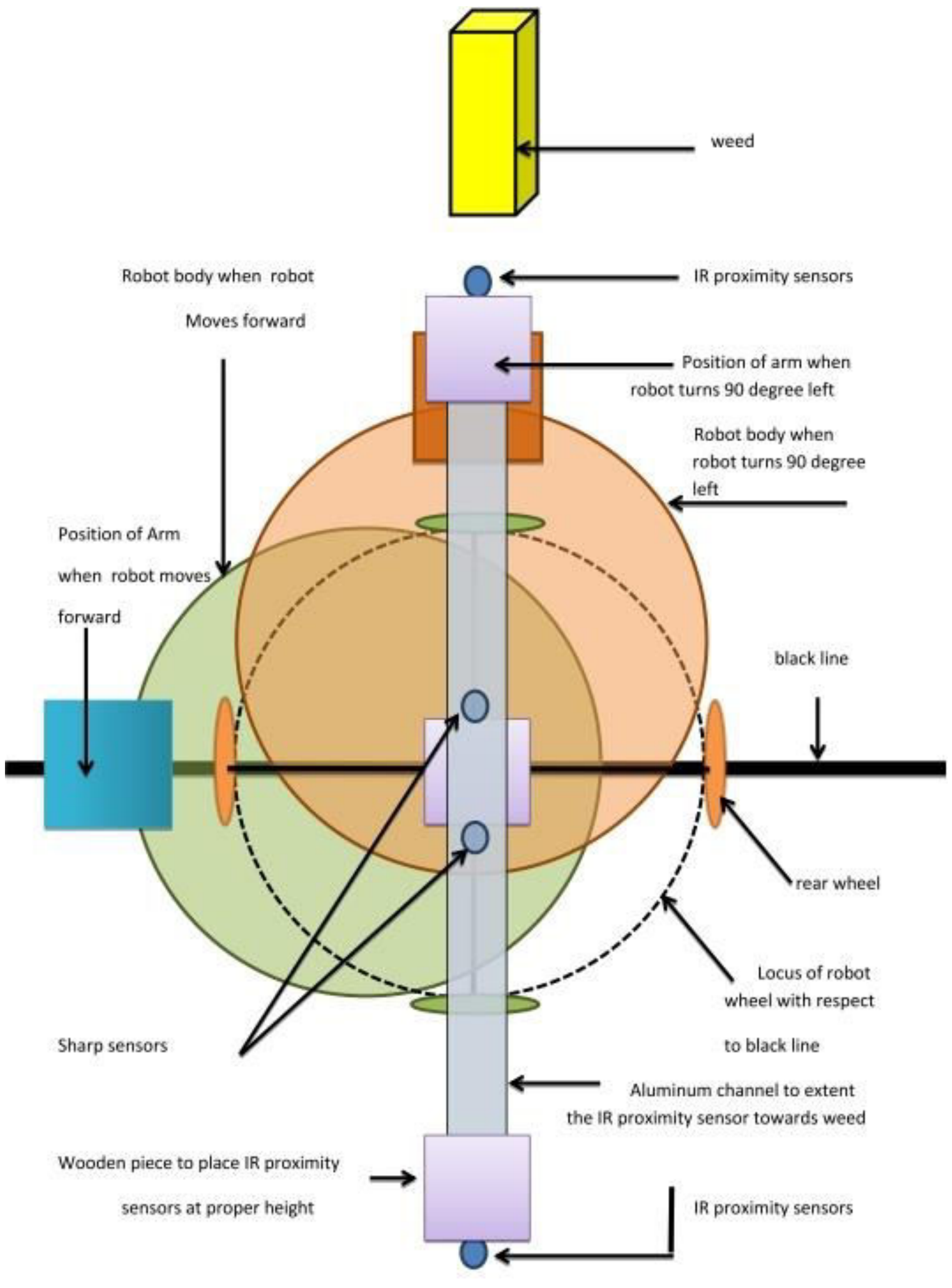


Fig: Front view of the sensor arrangement in the robot

The above diagram shows the arrangement of the Sharp and IR proximity sensors around the robot. Label 1 denotes the sharp sensor placed on the vertical projection near the rear wheel axis of the robot (ref: top view). Since sharp sensor is long range, the distance of 31 cm from the centre of robot to the centre of weed/plant is fine for sensing obstacles. The sharp sensor is placed such that it is below the 10 cm height above the trough to detect both weeds and plants.

Label 2 denotes the IR proximity sensors. Since these sensors are quite short-range, they are placed on suitable horizontal projections from the robot. Since the sensing ability varies approximately from 6-10 cm maximum, we place the IR proximity sensors at a safe distance of 5 cm from the weed/plant. The height of these sensors is adjusted between 10 cm and 15 cm above the level of the trough so that when it encounters a plant, sensor activates while if weed is encountered, sensor remains low. This phenomenon is illustrated in the diagram. For the plant on the left side, the IR proximity senses a reflection and goes high. On the other hand, for the weed on the right, the sensor detects no reflection and stays low.



<u>Fig: Top view of the sensor arrangement in the robot</u>

Q-3Teams have to prepare a gripper mechanism for uprooting the weeds.

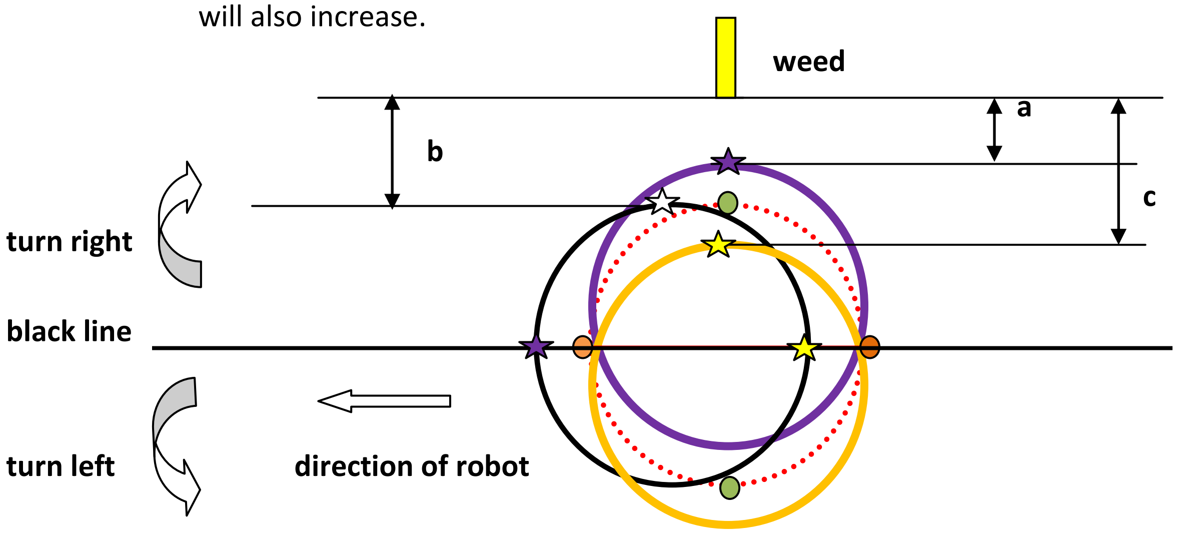
a) Choosean option you would like to use to position the gripper mechanism on the robot. (5)

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

Answer: _________Front_____________

The robot is supposed to turn left or right when a weed is detected. Now the front surface of the robot is towards the weed. The drawbacks of placing the gripping mechanism at other points are examined below:-

1. If we place at back, the length of arm will be large.
2. If we place at one side, length of arm will also be large. The additional problem is that we have to place sensor and gripper in a certain alignment to successfully uproot the plant due to shortage of available space when they are both putin one side.
3. If we place gripper on both sides, then number of motors is doubled and weight will also increase.



Clearly**a<b<c**

☆ = Position of gripper at one side

★ = Position of gripper at back

★ = Position of gripper at front

• • • • • Locus of rear wheel with respect to the black line while turning left or right

▬ Robot contour when turn left

▬ Robot contour when turn right

▬ Robot contour when moving forward

🟠 Position of rear wheel when robot turns left or right

🟢 Position of rear wheel when robot is moving in forward direction

The above schematic diagram illustrates the fact that the distance of the arm to the weed will be shortest if the arm is placed in front as opposed to the sides or the back of the robot.

So we place gripper at the front of the robot and it gives the following advantages:-

i) The length of the arm is shortest.

ii) There is no clash between the position of gripper and sensor which would occur if placed on side(s).

iii) By just rotating robot 90 degree left or right, it can access weed on both sides of the black line.

iv) Only 2 servo motors are needed to perform whole gripper and arm action.

v) After collecting weed, the main arm rotates 180 degrees and places the uprooted weed in a dumper, fixed at the back of the robot.

vi) The mass distribution is uniform with respect to its axis of symmetry, i.e.,black line.

b) Draw a diagram to show the grippermechanism and how it is mounted on the robot. (15)

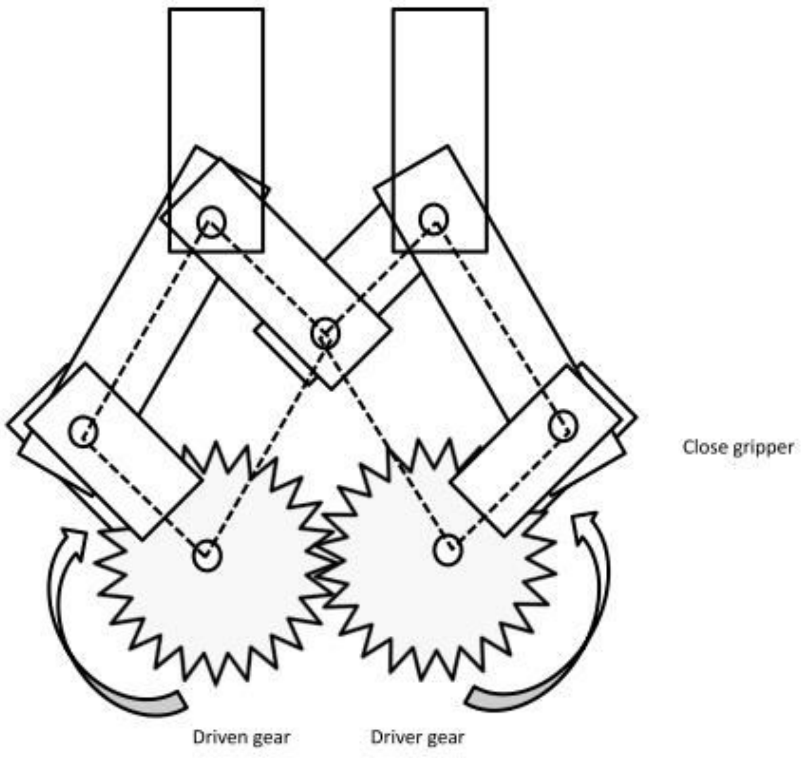


Fig: Gripper in closed position

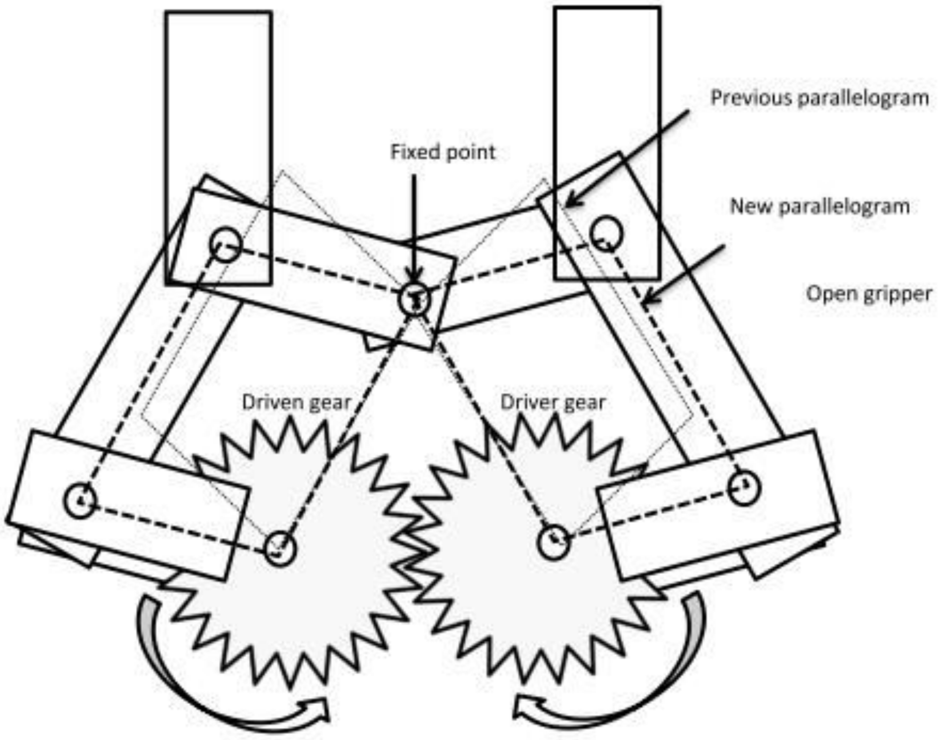
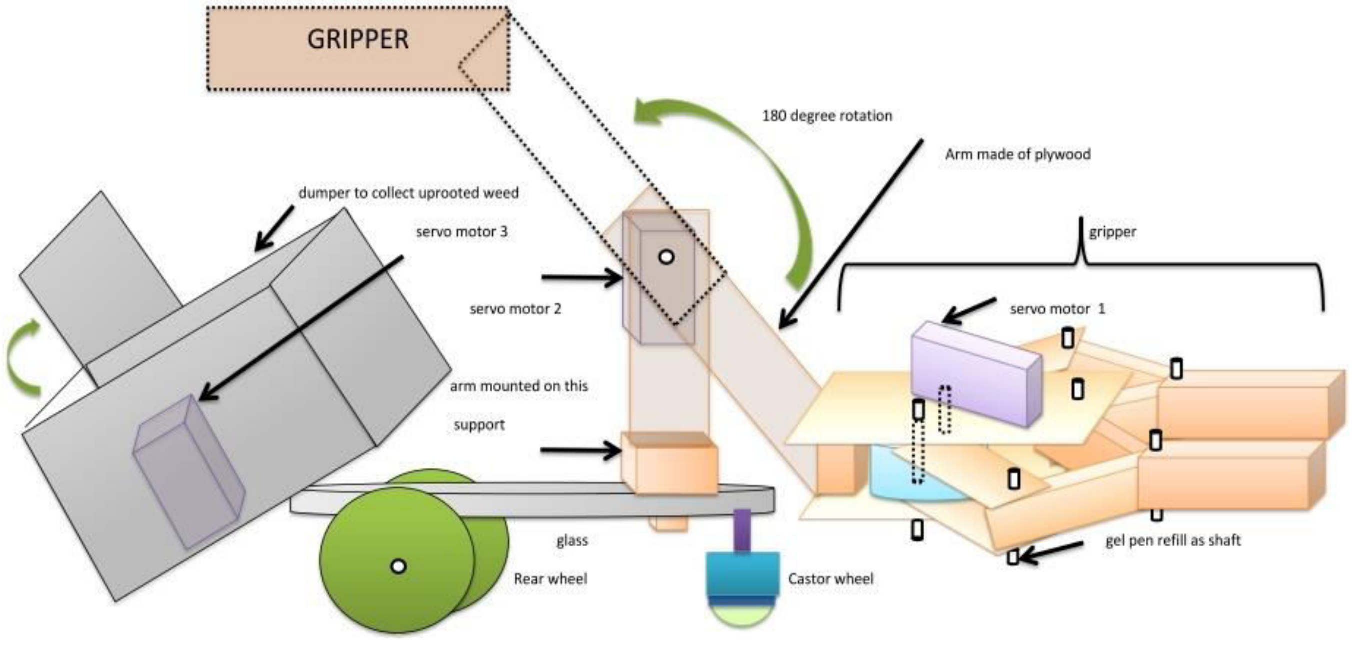


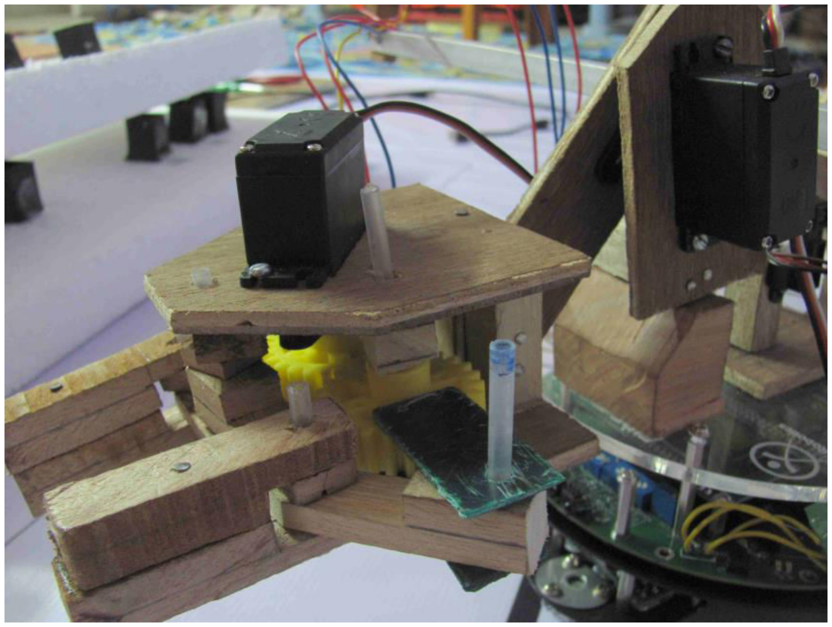
Fig: Gripper in open position(servo position zero)

Mechanism:The above schematic shows the operation of the gripper assembly. Here we use two identical plastic gears connected as shown in fig. The driving gear is connected to a servo motor which also actuates the driven gear. The driving and the driven gears rotate in opposite directions to move the two flat pieces towards or away from each other for gripping and releasing respectively. The degree of freedom (DOF) for our arm assembly is considered in the vertical plane and for the end-effecter, there is only provision for closing and opening of the gripper in a horizontal plane for uprooting of the weeds.

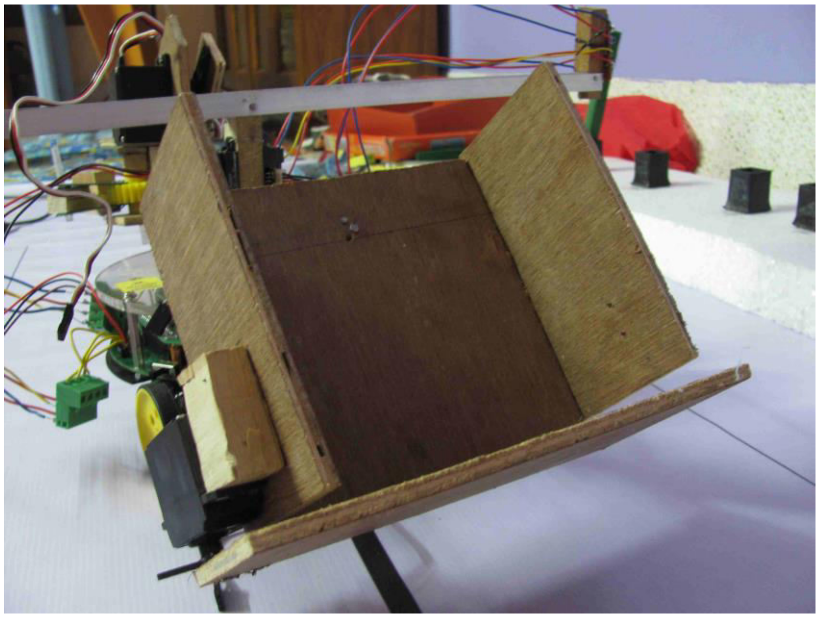


The robotic arm is mounted on the front of the robot as shown. After uprooting the weed with the gripper, the arm moves 180 degrees backwards as shown and drops the weed in the container. When the robot reaches the 2nd or 4th intersection, it dumps all the uprooted weeds in container into the deposition zone. The 3rd deposition zone is not used since there is not weed to be uprooted between the 2nd and 3rd corners.

The entire arm assembly is made with wood as shown below for light weight and easy flexibility.



<u>ARM AND GRIPPER ASSEMBLY</u>



<u>CONTAINER FOR COLLECTION OF WEEDS</u>

Q-4 Choose the actuator you will use to design the grippermechanism. (5)

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

Answer:________Servo motor__________

Justify your answer by stating the advantage of the chosen actuator over others. Also give reasons for not using other actuators.

We intend to use servo motors in our theme because it has the following characteristics:-

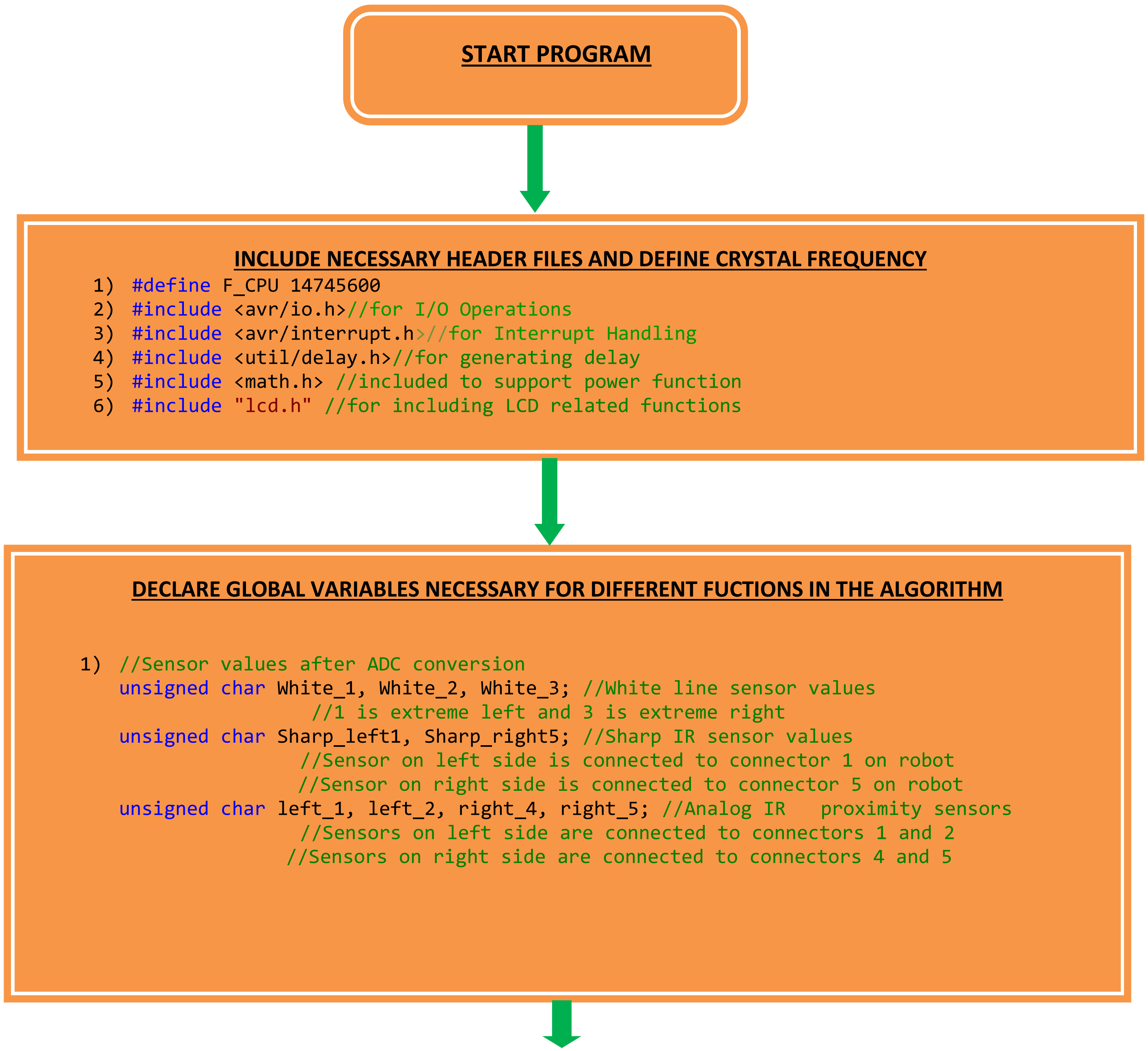
1. Servo motor produces high torque.
2.Servo motor has low inertia.
3. Size of the servo motor is also compact.
4. Speed-torque characteristics is linear which is required for fine speed control.
5. Since it is a closed loop system, position control is very precise through error actuated (negative feedback) correction mechanism.

On using DC motor, the power is delivered using PWM technique; then the motor doesnot receive full power. As a result of switching on and off the power frequently, then the motor suffers stuttering. Also they require a motor driver IC to interface them with the microcontroller. In case of FIREBIRD V, there is a provision for an extra L293D IC to interface DC motors but it is rather inconvenient.
However, for a servo motor, there is a DC motor coupled with a position sensing unit, a gear reduction unit, a control circuit. It doesnot rotate freely as DC motor, but rotates its shaft only by a finite angle so as to match the present position of the command shaft. The position correction mechanism helps the system to determine the final position of the shaft. PWM is used for the control signal of servo motors. However, unlike DC motors it's the duration of the positive pulse that determines the position, rather than speed, of the servo shaft.Since the FIREBIRD V has readymade servo connectors for interfacing servo motors with the robot, they are easy to operate.
In case of stepper motors,although they exhibit position control system but they have no feedback system for automatic correction. The lack of correction mechanism may lead to missed steps in a stepper motor. They too require the L293D IC for interfacing. Also they are more bulky.But servo motors have the feedback control system which is performed by encoder and controller to avoid the problem that could be caused by stepper motor.
So, due to this advantage and the previous mentioned characteristics, servo motors are the preferred option for gripper mechanism.

Algorithm Analysis

Q-1 Draw a flowchart illustrating the algorithm used to complete the entire task. (40)

The flowchart mentioned below gives a tentative algorithm in the sequence of the code of the Embedded C program. At first, the functions necessary for the main algorithm are described along with relevant details. Then the main algorithm is illustrated where the necessary functions are called to perform specific tasks to solve the weeding robot problem.



START PROGRAM

INCLUDE NECESSARY HEADER FILES AND DEFINE CRYSTAL FREQUENCY

```
1) #define F_CPU 14745600
2) #include <avr/io.h>//for I/O Operations
3) #include <avr/interrupt.h>//for Interrupt Handling
4) #include <util/delay.h>//for generating delay
5) #include <math.h> //included to support power function
6) #include "lcd.h" //for including LCD related functions
```

DECLARE GLOBAL VARIABLES NECESSARY FOR DIFFERENT FUCTIONS IN THE ALGORITHM

```
1) //Sensor values after ADC conversion
   unsigned char White_1, White_2, White_3; //White line sensor values
                    //1 is extreme left and 3 is extreme right
   unsigned char Sharp_left1, Sharp_right5; //Sharp IR sensor values
                   //Sensor on left side is connected to connector 1 on robot
                   //Sensor on right side is connected to connector 5 on robot
   unsigned char left_1, left_2, right_4, right_5; //Analog IR   proximity sensors
                   //Sensors on left side are connected to connectors 1 and 2
                  //Sensors on right side are connected to connectors 4 and 5
```

DECLARE GLOBAL VARIABLES NECESSARY FOR DIFFERENT FUCTIONS IN THE ALGORITHM (CONTD)

```
2)  //Thresholds for sensor values
    unsigned int tw_1;//Threshold for white line sensor 1
    unsigned int tw_2;//Threshold for white line sensor 2
    unsigned int tw_3;//Threshold for white line sensor 3
    unsigned int ts_1;//Threshold for sharp sensor 1
    unsigned int ts_5;//Threshold for sharp sensor 5
    unsigned int tr_1;//Threshold for analog IR sensor 1
    unsigned int tr_2;//Threshold for analog IR sensor 2
    unsigned int tr_4;//Threshold for analog IR sensor 4
    unsigned int tr_5;//Threshold for analog IR sensor 5

3)  //Arena monitoring variables
    unsigned int count = 0; //total no of weeds uprooted
    unsigned int weed_full = 0;//a boolean variable to check if container contains
                               //uprooted weeds. If 0, no weed present.
                               //If 1, at least one uprooted weed present in container.

4)  //Variables for black line navigation
    float pGain = k;//gain factor for proportional control, k is calibrated accordingly
    float control; //control variable
    float s; //average value for white line sensors
    unsigned int white_value;//the contents of all 3 white line sensor values
                    //arranged sequentially as a single integer variable


5)  //Position encoder usage variables
    unsigned long int ShaftCountLeft = 0; //to keep track of left position encoder
    unsigned long int ShaftCountRight = 0; //to keep track of right position encoder
    unsigned int Degrees; //to accept angle in degrees for turning
```

CONFIGURING PORTS AND PINS IN THE MASTER MICRO-CONTROLLER FOR VARIOUS FUNCTIONALITIES

```
1)  //Configure PORTB 5 pin for servo motor 1 operation
    void servo1_pin_config (void)
    {
           DDRB = DDRB | 0x20; //making PORTB 5 pin output
           PORTB = PORTB | 0x20; //setting PORTB 5 pin to logic 1
    }

    //Configure PORTB 6 pin for servo motor 2 operation
    void servo2_pin_config (void)
    {
           DDRB = DDRB | 0x40; //making PORTB 6 pin output
           PORTB = PORTB | 0x40; //setting PORTB 6 pin to logic 1
    }

    //Configure PORTB 7 pin for servo motor 3 operation
    void servo3_pin_config (void)
    {
           DDRB = DDRB | 0x80; //making PORTB 7 pin output
           PORTB = PORTB | 0x80; //setting PORTB 7 pin to logic 1
    }
```

CONFIGURING PORTS AND PINS IN THE MASTER AND SLAVE MICRO-CONTROLLER FOR VARIOUS FUNCTIONALITIES (CONTD)

```
2) //Configure LCD port which is needed to display values on the LCD
   void lcd_port_config (void)
   {
          DDRC = DDRC | 0xF7; //all the LCD pin's direction set as output
          PORTC = PORTC & 0x80; // all the LCD pins are set to logic 0 except PORTC 7
   }

3) //ADC pin configuration which is needed to convert analog sensor values to digital
   void adc_pin_config (void)
   {
          DDRF = 0x00; //set PORTF direction as input
          PORTF = 0x00; //set PORTF pins floating
          DDRK = 0x00; //set PORTK direction as input
          PORTK = 0x00; //set PORTK pins floating
   }


4) //Configure ports to enable robot's motion
   void motion_pin_config (void)
   {
          DDRA = DDRA | 0x0F;
          PORTA = PORTA & 0xF0;
          DDRL = DDRL | 0x18;   //Setting PL3 and PL4 pins as output for PWM generation
          PORTL = PORTL | 0x18; //PL3 and PL4 pins are for velocity control using PWM.
   }

5) //Configure Buzzer. Buzzer is turned ON when robot reaches END point in the arena.
   void buzzer_pin_config (void)
   {
          DDRC = DDRC | 0x08;//Setting PORTC 3 as output
          PORTC = PORTC & 0xF7;//Setting PORTC 3 logic low to turnoff buzzer
   }

6) //Configure INT4 (PORTE 4) pin as input for the left position encoder
   void left_encoder_pin_config (void)
   {
          DDRE  = DDRE & 0xEF;  //Set the direction of the PORTE 4 pin as input
          PORTE = PORTE | 0x10; //Enable internal pullup for PORTE 4 pin
   }

7) //Configure INT5 (PORTE 5) pin as input for the right position encoder
   void right_encoder_pin_config (void)
   {
          DDRE  = DDRE & 0xDF;  //Set the direction of the PORTE 5 pin as input
          PORTE = PORTE | 0x20; //Enable internal pullup for PORTE 5 pin
   }
```

ENABLE POSITION ENCODER INTERRUPTS

```
    //Position encoders are useful while traversing the grid in the white line
    //following part. The actual procedure for traversing the grid is described
    //later

1)  //Initializes the left position encoder interrupt.
    void left_position_encoder_interrupt_init (void) //Interrupt 4 enable
    {
           cli(); //Clears the global interrupt
           EICRB = EICRB | 0x02; // INT4 is set to trigger with falling edge
           EIMSK = EIMSK | 0x10; // Enable Interrupt INT4 for left position
                                 //encoder
           sei();   // Enables the global interrupt
    }

2)  //Initializes the right position encoder interrupt
    void right_position_encoder_interrupt_init (void) //Interrupt 5 enable
    {
           cli(); //Clears the global interrupt
           EICRB = EICRB | 0x08; // INT5 is set to trigger with falling edge
           EIMSK = EIMSK | 0x20; // Enable Interrupt INT5 for right position
                                 //encoder
           sei();   // Enables the global interrupt
    }
```

INTERRUPT SERVICE ROUTINE (ISR) DEFINATION

```
1)  //ISR for left position encoder
    ISR(INT4_vect)
    {
           ShaftCountLeft++;//increment left shaft position count
    }

2)  //ISR for right position encoder
    ISR(INT5_vect)
    {
           ShaftCountRight++;//increment right shaft position count
    }
```

INTIALISATION OF PORTS, PINS AND CHANNELS

1)
```
//This function initializes the ports and pins already configured before
void port_init()
{
        lcd_port_config();
        adc_pin_config();
        servo1_pin_config(); servo2_pin_config(); servo3_pin_config();
        motion_pin_config();
        buzzer_pin_config();
        left_encoder_pin_config();
        right_encoder_pin_config();
}
```

2)
```
// Timer 5 initialized in PWM 8 bit fast mode needed for velocity control
// Prescale:256, TOP=0x00FF, Timer Frequency:225.000Hz
void timer5_init()
{
        TCCR5B = 0x00;//Stop
        TCNT5H = 0xFF;//Counter higher 8-bit value
        TCNT5L = 0x01;//Counter lower 8-bit value
        OCR5AH = 0x00;//Output compare register high value for Left Motor
        OCR5AL = 0xFF;//Output compare register low value for Left Motor
        OCR5BH = 0x00;//Output compare register high value for Right Motor
        OCR5BL = 0xFF;//Output compare register low value for Right Motor
        OCR5CH = 0x00;//Output compare register high value for Motor C1
        OCR5CL = 0xFF;//Output compare register low value for Motor C1
        TCCR5A = 0xA9;//COM5A1=1,COM5A0=0;COM5B1=1,COM5B0=0;COM5C1=1, COM5C0=0
        TCCR5B = 0x0B;//WGM12=1; CS12=0, CS11=1, CS10=1 (Prescaler=64)
}
```

3)
```
//TIMER1 initialization in 10 bit fast PWM mode prescale:256 PWM 10bit fast,
//TOP=0x03FF ,actual value: 52.25Hz ; needed for servo operation
void timer1_init(void)
{
 TCCR1B = 0x00; //stop
 TCNT1H = 0xFC; //Counter high value to which OCR1xH value is to be compared
 TCNT1L = 0x01;//Counter low value to which OCR1xH value is to be compared
 OCR1AH = 0x03;      //Output compare Register high value for servo 1
 OCR1AL = 0xFF;      //Output Compare Register low Value For servo 1
 OCR1BH = 0x03;      //Output compare Register high value for servo 2
 OCR1BL = 0xFF;      //Output Compare Register low Value For servo 2
 OCR1CH = 0x03;      //Output compare Register high value for servo 3
 OCR1CL = 0xFF;      //Output Compare Register low Value For servo 3
 ICR1H  = 0x03;
 ICR1L  = 0xFF;
 TCCR1A = 0xAB; /*{COM1A1=1, COM1A0=0; COM1B1=1, COM1B0=0; COM1C1=1 COM1C0=0}
   For Overriding normal port functionality to OCRnA outputs.
  {WGM11=1, WGM10=1} Along With WGM12 in TCCR1B for Selecting FAST PWM Mode*/
 TCCR1C = 0x00;
 TCCR1B = 0x0C; //WGM12=1; CS12=1, CS11=0, CS10=0 (Prescaler=256)
}
```

4)
```
//Function to Initialize ADC Channels
void adc_init()
{
        ADCSRA = 0x00;
        ADCSRB = 0x00;//MUX5 = 0
        ADMUX = 0x20;//Vref=5V external --- ADLAR=1 --- MUX4:0 = 0000
        ACSR = 0x80;
        ADCSRA = 0x86;//ADEN=1 --- ADIE=1 --- ADPS2:0 = 1 1 0
}
```

DEFINING BUZZER FUNCTIONS

1)
```
//Function to turn ON the buzzer
void buzzer_on (void)
{
        unsigned char port_restore = 0;
        port_restore = PINC;
        port_restore = port_restore | 0x08;
        PORTC = port_restore;
}
```

2)
```
//Function to turn OFF the buzzer
void buzzer_off (void)
{
        unsigned char port_restore = 0;
        port_restore = PINC;
        port_restore = port_restore & 0xF7;
        PORTC = port_restore;
}
```

MOTOR DIRECTION AND VELOCITY CONTROL FUNCTIONS

1)
```
//Function used for setting motor's direction
void motion_set (unsigned char Direction)
{
        unsigned char PortARestore = 0;

        Direction &= 0x0F;//removing upper nibble for protection
        PortARestore = PORTA;//reading PORTA original status
        PortARestore &= 0xF0;//making lower direction nibble to 0
        PortARestore |= Direction;//adding lower nibble for forward command
                                  //and restoring the PORTA status
        PORTA = PortARestore;//executing the command
}
//In the above function, different hexcodes as arguments set corresponding
//directions for locomotion as follows:-
//motion_set (0x06) – FORWARD,i.e, both wheels forward
//motion_set(0x04) – SOFT LEFT,i.e, left wheel stationary & right forward
//motion_set(0x02) - SOFT RIGHT,i.e, left wheel forward & right stationary
//motion_set(0x05) – LEFT,i.e, left wheel backward & right forward
//motion_set(0x0A) – RIGHT,i.e, left wheel forward & right backward
//motion_set(0x09) – BACK,i.e, both wheels backward
//motion_set (0x00) – STOP,i.e, both wheels stationary
//Accordingly, functions can be defined for each motion separately,
//e.g – forward(), left(), etc.
```

2)
```
//Function for velocity control using PWM by Timer 5
void velocity (unsigned char left_motor, unsigned char right_motor)
{
        OCR5AL = (unsigned char)left_motor;
        OCR5BL = (unsigned char)right_motor;
        //Since timer 5 has maximum value of 255, its OCR5AL and OCR5BL
        //registers cannot exceed that value. Hence maximum velocity of
        //motor corresponds to the value of 255 in the registers. Also,
        //for practical cases, a value less than 100 is too low to actuate
        //the motor. So the values assigned must be within 100 to 255.
}
```

FIXED LOCOMOTION FUNCTIONS

```
//These functions are used for moving the robot forward for backward by a
//certain specified distance, turning left or right by certain degrees, etc.
//These functions may be utilized while traversing the grid of white lines.
```

1)
```
//Function used for moving robot forward or backward by specified distance
void linear_distance_mm(unsigned int DistanceInMM)
{
       float ReqdShaftCount = 0;
       unsigned long int ReqdShaftCountInt = 0;
       ReqdShaftCount = DistanceInMM / 5.338; //division by resolution to get
                                              //shaft count
       ReqdShaftCountInt = (unsigned long int) ReqdShaftCount;
       ShaftCountRight = 0;
       while(1)
       {
              //the lcd_print function is defined in the “lcd.h” header file.
              //Here the three variables below may be displayed so as check
              //if the position encoder interrupts are working fine or not.
              //During final run, they may be by-passed.
              lcd_print(1, 1, ShaftCountLeft, 3);
              lcd_print(1, 5, ShaftCountRight, 3);
              lcd_print(1, 9, ReqdShaftCountInt, 3);
              if(ShaftCountRight > ReqdShaftCountInt)//If right(or left) shaft
                                                    //exceeds the required shaft count
              {
                     break;
               }
        }
        stop(); //Stop action
}
```

2)
```
//Function used for turning robot left or right by specified degrees.
void angle_rotate(unsigned int Degrees)
{
       float ReqdShaftCount = 0;
       unsigned long int ReqdShaftCountInt = 0;
       ReqdShaftCount = (float) Degrees/ 4.090; //division by resolution to
                                                //get shaft count
        ReqdShaftCountInt = (unsigned int) ReqdShaftCount;
        ShaftCountRight = 0;
        ShaftCountLeft = 0;
        while (1)
       {
              lcd_print(1, 1, ShaftCountLeft, 3);
              lcd_print(1, 5, ShaftCountRight, 3);
              lcd_print(1, 9, ReqdShaftCountInt, 3);
              if((ShaftCountRight >= ReqdShaftCountInt) | (ShaftCountLeft >=
                  ReqdShaftCountInt))
                     break;
        }
       stop(); //Stop action
}

//The “angle_rotate” function may be implemented in normal left/right
//mode or soft left/right mode. In normal mode, there are 88 pulses for 360
//degrees rotation,i.e, 4.090 degrees per count. But in soft mode, there are
//176 pulses for 360 degrees rotation,i.e, 2.045 degrees per count. So in soft
//mode, if x degrees turning is to be achieved actually, 2*x must be passed as
//argument to the “angle_rotate” function since only one wheel is moving.
```

FUNCTION FOR ADC CONVERSION

1)
```
//The 3 white line sensors, 2 sharp sensors and 4 IR analog proximity sensors
//(used in our theme) connected to ATmega2560 master obtain their ADC values
//from this function which accepts the Channel Number and returns the
//corresponding value
unsigned char ADC_Conversion(unsigned char Ch)
{
        unsigned char a;
        if(Ch>7)
        {
                ADCSRB = 0x08;
        }
        Ch = Ch & 0x07;
        ADMUX= 0x20| Ch;
        ADCSRA = ADCSRA | 0x40;//Set start conversion bit
        while((ADCSRA&0x10)==0);//Wait for ADC conversion to complete
        a=ADCH;
        ADCSRA = ADCSRA|0x10;//clear ADIF(ADC Interrupt Flag) by writing 1 to it
        ADCSRB = 0x00;
        return a;
}
```

FUNCTIONS FOR SERVO MOTOR CONTROL

```
//These functions control the servo motor rotation.

//Function to rotate Servo 1 by a specified angle in the multiples of 1.86
degrees
void servo_1(unsigned char degrees)
{
        float PositionPanServo = 0;
        PositionPanServo = ((float)degrees / 1.86) + 35.0;
        OCR1AH = 0x00;
        OCR1AL = (unsigned char) PositionPanServo;
}

//Function to rotate Servo 2 by a specified angle in the multiples of 1.86
degrees
void servo_2(unsigned char degrees)
{
        float PositionTiltServo = 0;
        PositionTiltServo = ((float)degrees / 1.86) + 35.0;
        OCR1BH = 0x00;
        OCR1BL = (unsigned char) PositionTiltServo;
}

//Function to rotate Servo 3 by a specified angle in the multiples of 1.86
degrees
void servo_3(unsigned char degrees)
{
        float PositionServo = 0;
        PositionServo = ((float)degrees / 1.86) + 35.0;
        OCR1CH = 0x00;
        OCR1CL = (unsigned char) PositionServo;
}
```

FUNCTION TO COMPUTE SENSOR VALUES

```
float readSensors()
{
        float avgSensors = 0.0;//weighted mean value of the white line sensors

        //Thresholding all Sharp and analog IR sensor values
        Sharp_left1 = (ADC_Conversion(9) > ts_1)? 1: 0;
        Sharp_right5 = (ADC_Conversion(13) > ts_5)? 1: 0;
        left_1 = (ADC_Conversion(4) > tr_1)? 1: 0;
        left_2 = (ADC_Conversion(5) > tr_2)? 1: 0;
        right_4 = (ADC_Conversion(7) > tr_4)? 1: 0;
        right_5 = (ADC_Conversion(8) > tr_5)? 1: 0;

        //Thresholding white line sensor values
        White_1 = (ADC_Conversion(3) > tw_1)? 1: 0;
        White_2 = (ADC_Conversion(2) > tw_2)? 1: 0;
        White_3 = (ADC_Conversion(1) > tw_3)? 1: 0;

        //calculation of weighted mean white line sensor value
        white_value = White_1*100 + White_2*10 + White_3;
        float sum = White_1 + White_2 + White_3;
        if(sum == 0.0) //prevent division by zero
        {
                return 0.0;
        }

        avgSensors = (White_1 * 1 + White_2 * 2 + White_3 * 3)/sum;
        return avgSensors;
}
```

FUNCTION TO GENERATE CONTROL VARIABLE ACCORDING TO PID ALGORITHM

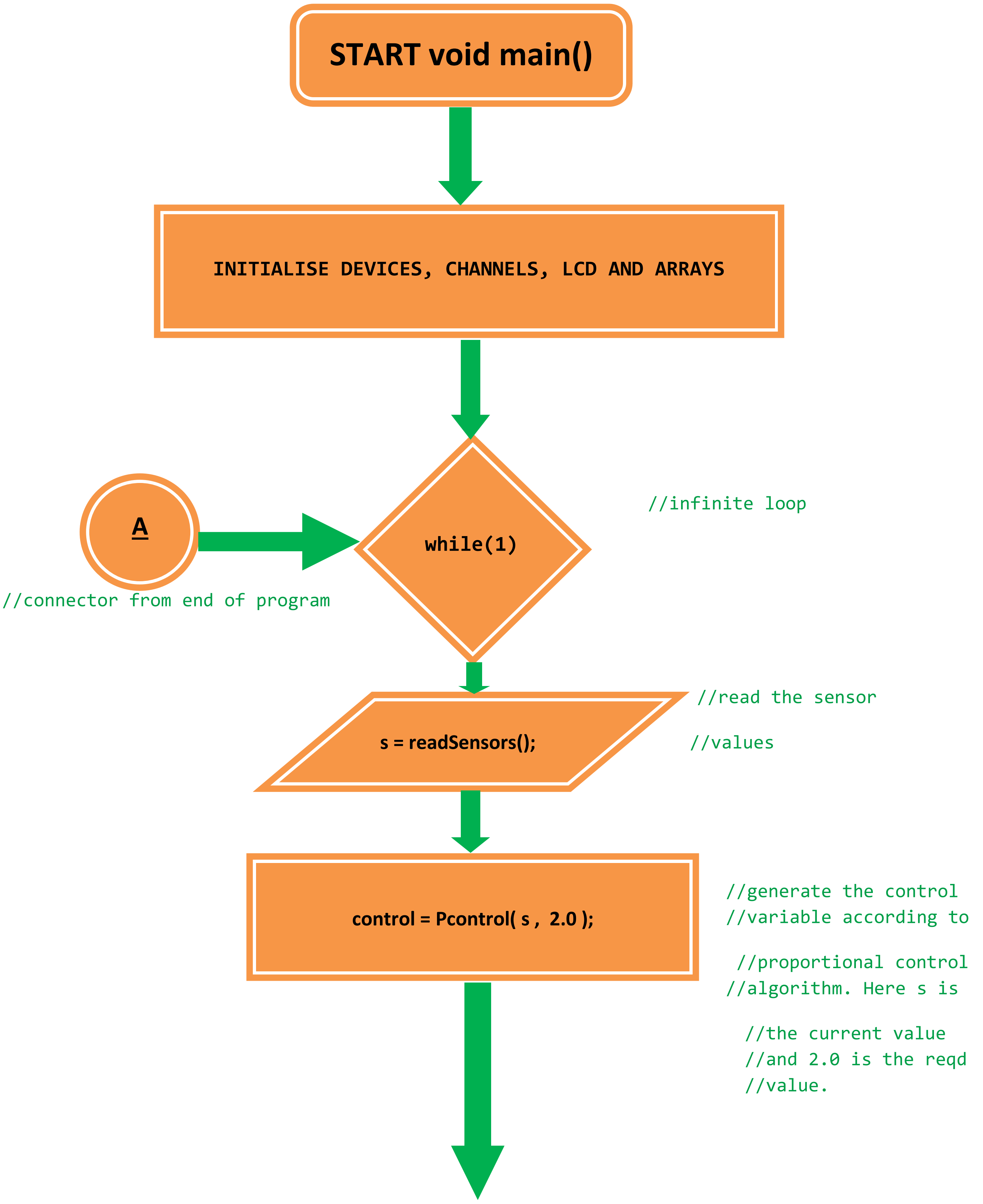
```
//Function to generate control variable using proportional control algorithm
float Pcontrol(float cur_value, float req_value) //Here the average sensor value
is the cur_value
//and 2.0 is the required value
{
        float con = 0.0;
        float error;
        error = req_value - cur_value;
        con = (pGain * error);
        if(con < 0) con *= -1; //To extract only magnitude of control without sign
        return con;
}
```

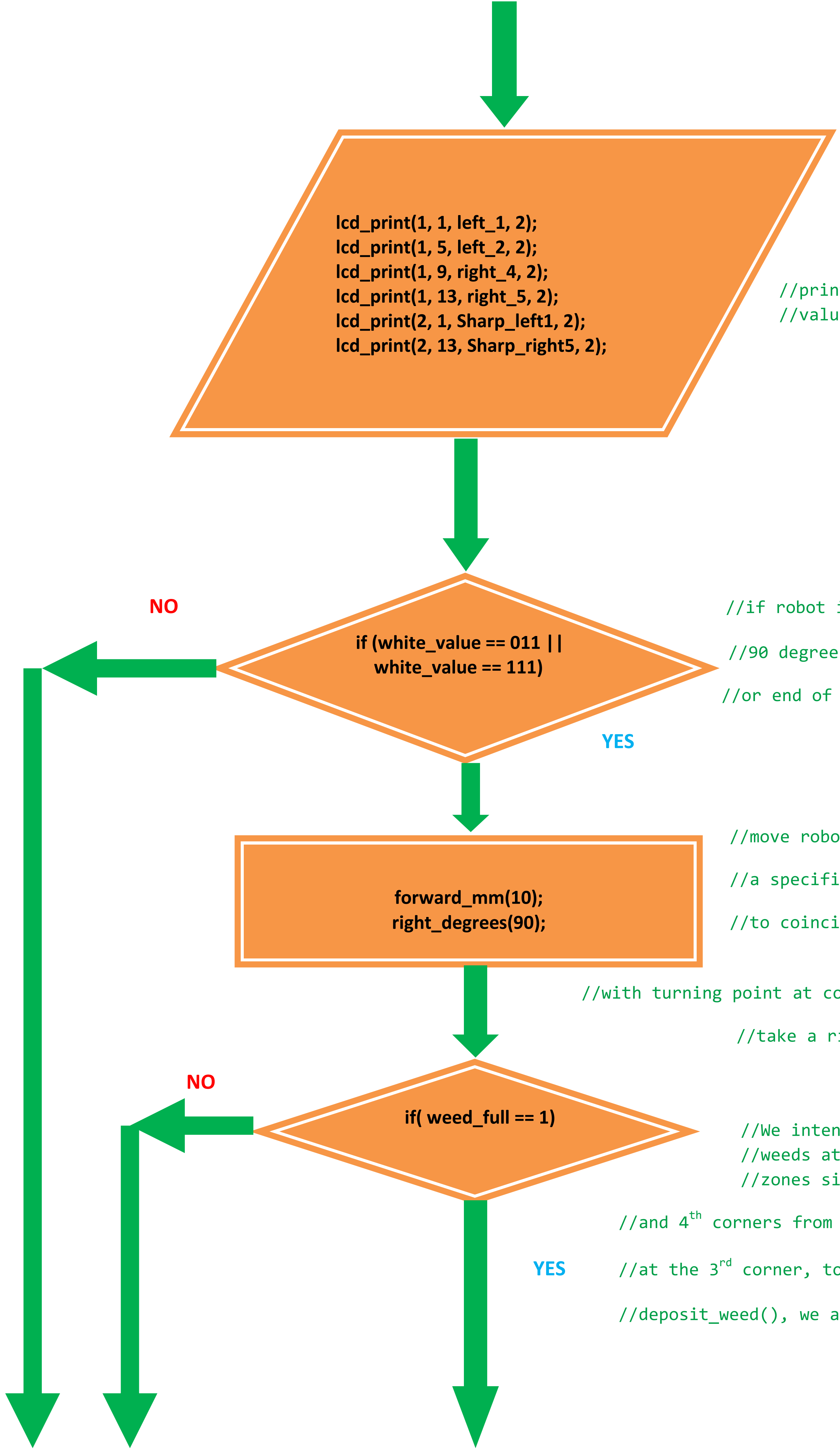
FUNCTIONS TO UPROOT AND DEPOSIT WEEDS

1)
```
//Function to uproot weed, Servo connector 1 is used to control the arm
//Servo connector 2 is used to control the gripper
void uproot_weed()
{
    //Initially bringing both servos to their zero position
    servo_1(0);
    servo_2(0);
    //moving arm downward
    for( int j = 0; j <= 180; j++)
    {
            servo_1(j);
            _delay_ms(10);
    }
    //gripping weed
    for(int k = 0; k < 75; k++)
    {
            servo_2(k);
            _delay_ms(10);
    }
    //moving arm upward
    for(int i = 180; i >= 0; i--)
    {
            servo_1(i);
            _delay_ms(10);
    }
    //releasing weed
    for(int l = 75; l >= 0; l--)
    {
            servo_2(l);
            _delay_ms(10);
    }

    count++; //increment no of weeds uprooted
    weed_full = 1;//the container now has at least 1 weed.
}
```

2)
```
//Function to open the container to drop weeds in deposition zone
//Servo 3 connector controls the servo used for this purpose
void deposit_weed()
{
    servo_3(0); //initial position is zero
    back_mm(20);//move robot 20 cm back into deposition zone

     //open container to let weeds fall
    for(int m = 0; m <= 140; m++)
    {
            servo_3(m);
            _delay_ms(10);
    }
    forward_mm(20); //move robot 20 cm forward to land up again over black line

    //close container
    for(int n = 140; n >= 0; n--)
    {
            servo_3(n);
            _delay_ms(10);
    }
    weed_full = 0;
}
```

THE MAIN ALGORITHM CORRESPONDING TO THE MAIN BLOCK IN THE EMBEDDED C PROGRAM





```
lcd_print(1, 1, left_1, 2);
lcd_print(1, 5, left_2, 2);
lcd_print(1, 9, right_4, 2);
lcd_print(1, 13, right_5, 2);
lcd_print(2, 1, Sharp_left1, 2);
lcd_print(2, 13, Sharp_right5, 2);
```

```
//print sensor
//values on LCD
```

```
if (white_value == 011 ||
white_value == 111)
```

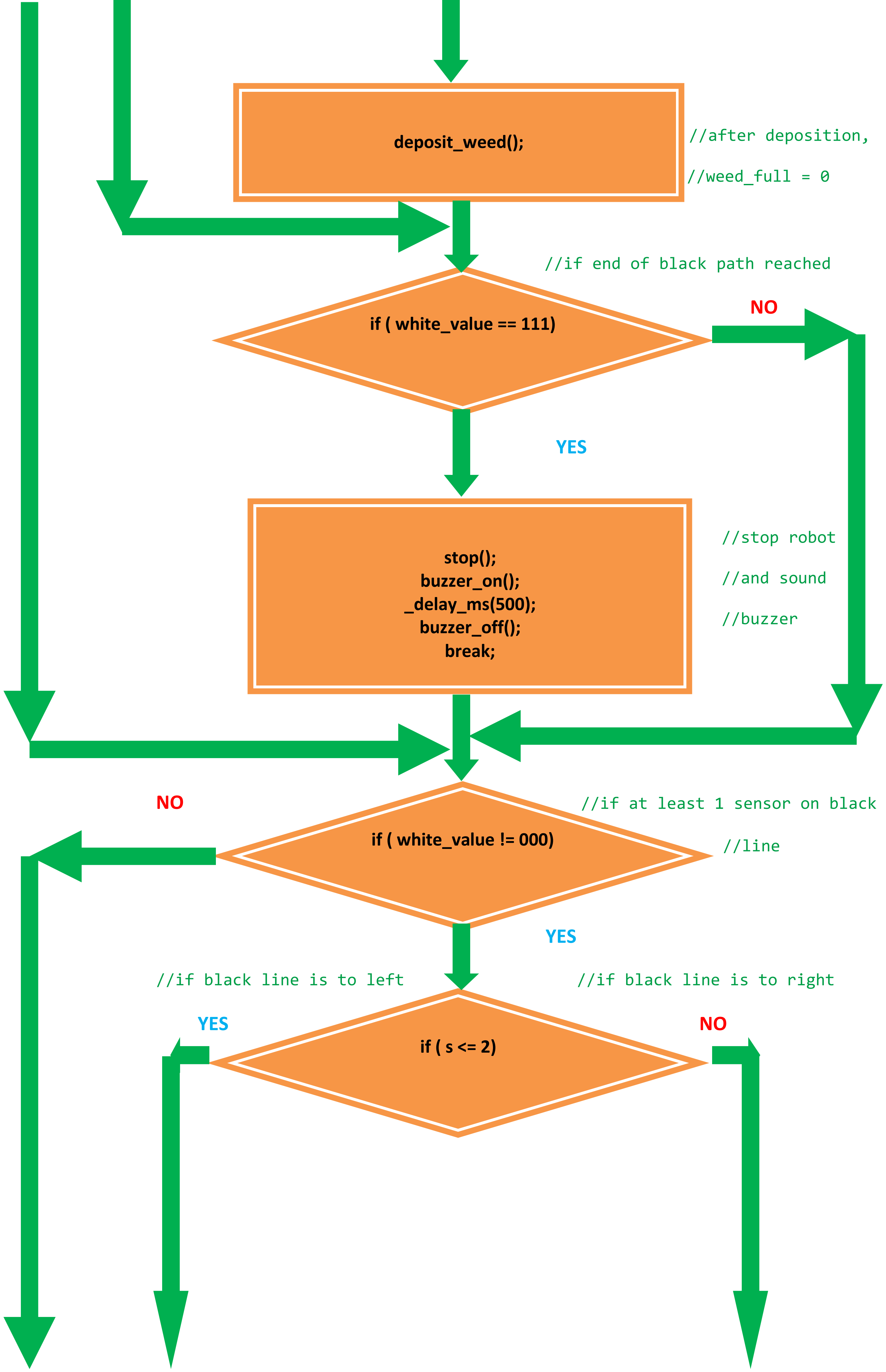
```
//if robot is at the
//90 degree right turns
//or end of black path
```

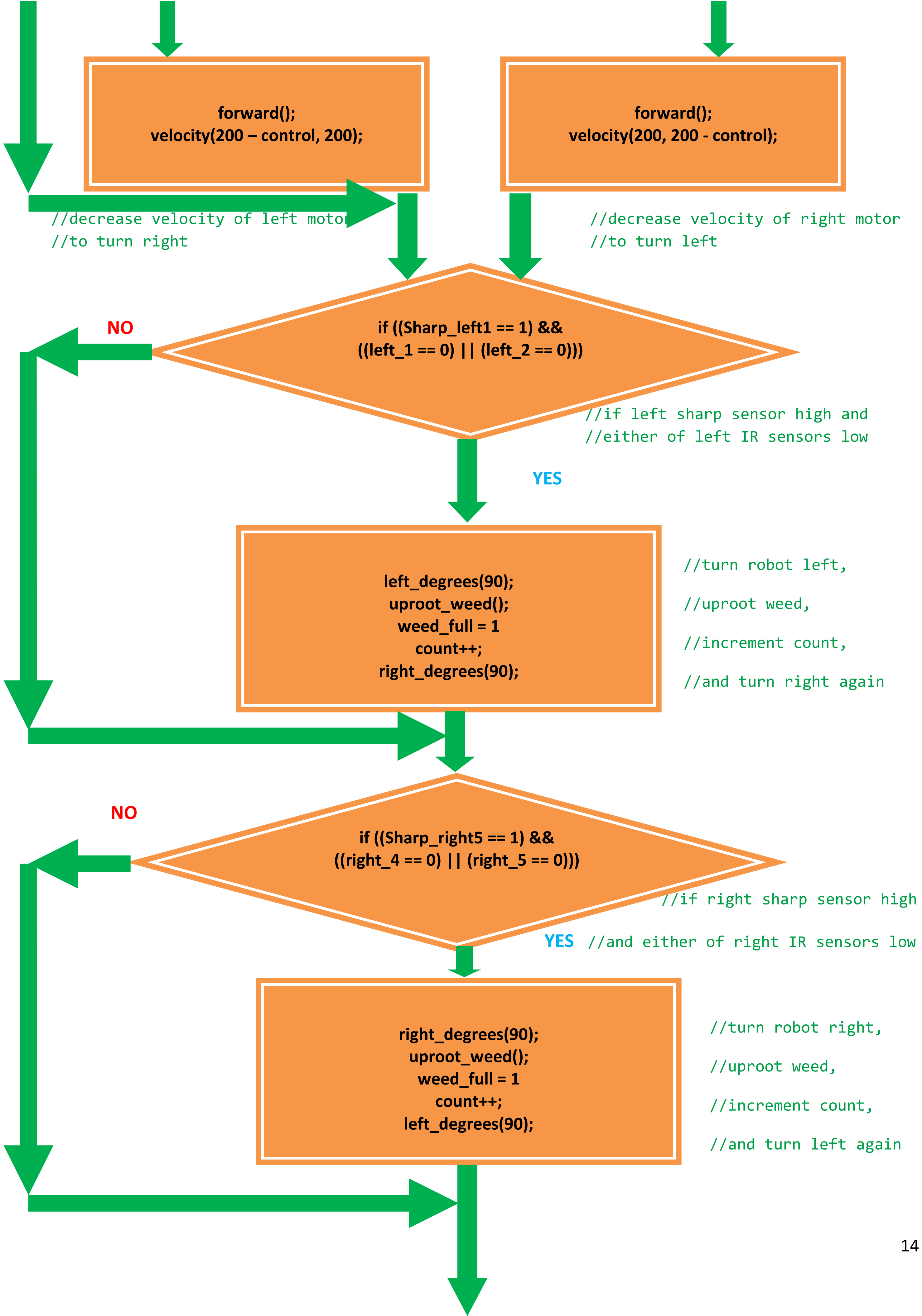
```
forward_mm(10);
right_degrees(90);
```

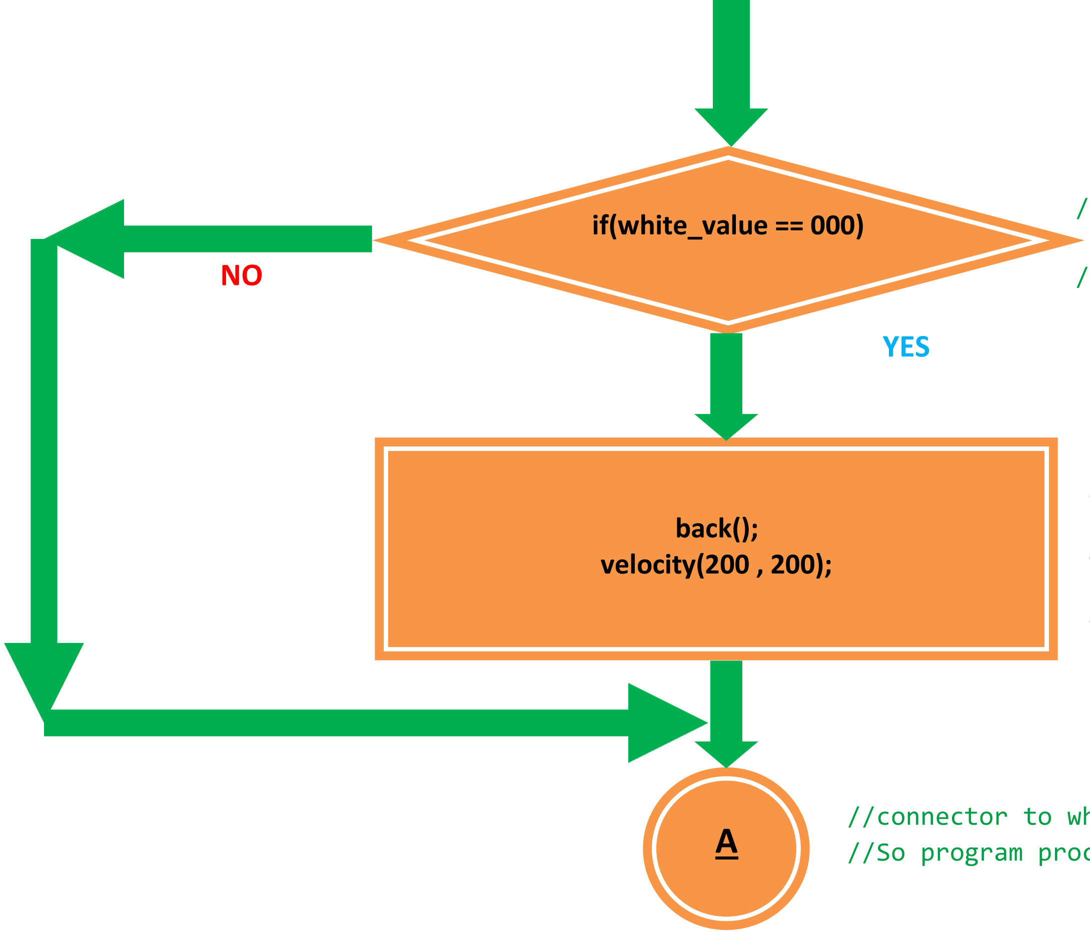
```
//move robot forward by
//a specified distance
//to coincide wheel axis
//with turning point at corner and then
//take a right turn
```

```
if( weed_full == 1)
```

```
//We intend to drop
//weeds at the deposition
//zones situated at 2nd
//and 4th corners from start. However
//at the 3rd corner, to restrict
//deposit_weed(), we apply this check
```


deposit_weed();
//after deposition,
//weed_full = 0
//if end of black path reached
if (white_value == 111)
NO
YES
stop();
buzzer_on();
_delay_ms(500);
buzzer_off();
break;
//stop robot
//and sound
//buzzer
NO
//if at least 1 sensor on black
//line
if (white_value != 000)
YES
//if black line is to left
//if black line is to right
YES
NO
if (s <= 2)


forward();
velocity(200 – control, 200);
forward();
velocity(200, 200 - control);
//decrease velocity of left motor
//to turn right
//decrease velocity of right motor
//to turn left
if ((Sharp_left1 == 1) &&
((left_1 == 0) || (left_2 == 0)))
NO
//if left sharp sensor high and
//either of left IR sensors low
YES
left_degrees(90);
uproot_weed();
weed_full = 1
count++;
right_degrees(90);
//turn robot left,
//uproot weed,
//increment count,
//and turn right again
if ((Sharp_right5 == 1) &&
((right_4 == 0) || (right_5 == 0)))
NO
//if right sharp sensor high
YES
//and either of right IR sensors low
right_degrees(90);
uproot_weed();
weed_full = 1
count++;
left_degrees(90);
//turn robot right,
//uproot weed,
//increment count,
//and turn left again



```
//if no sensor on black

//line

//move robot back till

//sensor again detects

//black line

//connector to while(1) condition before
//So program proceeds to next iteration
```