DEVELOPMENT OF ROBOTIC OPERATING SYSTEM (ROS)

4.1. Diagram of the Robotic Operating System

The Robotic Operating System developed for Pony contains drivers for Wireless data transmission, Motion control, Display, Collision Avoidance, Landmark detection, etc. These drivers communicate with the Destination Station Command Receiver Application and the hardware of the robot as shown in the Figure 4.1.

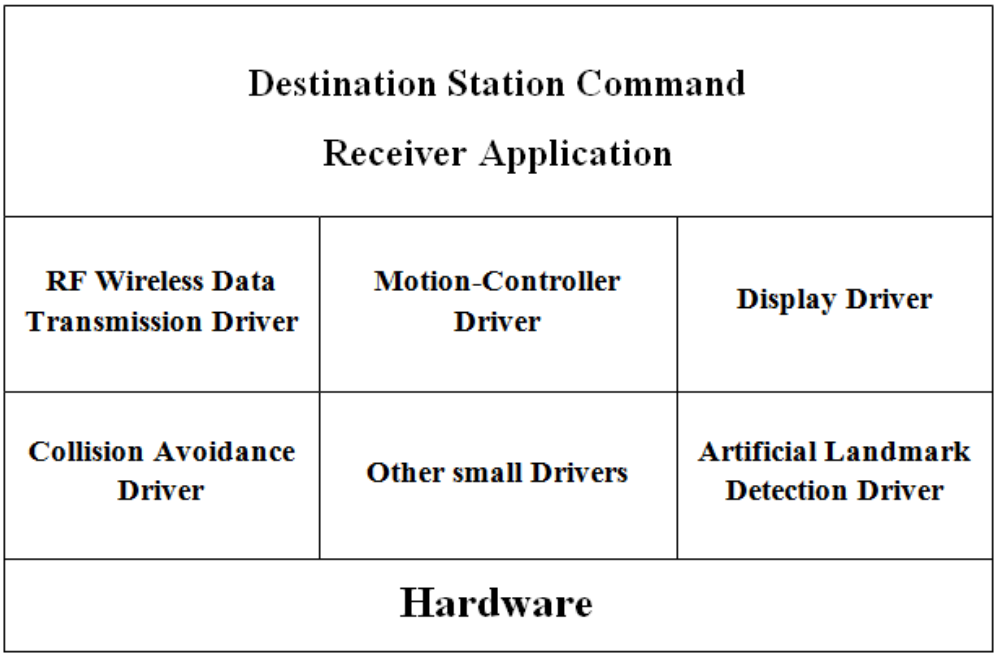

Figure 4.1: Block Diagram of the Robotic Operating System

4.2. Development of the Drivers of the ROS

4.2.1. RF Data Transmission and Reception Driver

The RF Data Transmission and Reception Driver transmits serial synchronous data wirelessly. The data is then received by the robot as a command and performs computations accordingly. The code below describes the vital working of sending the clock pulse along with the data pulse for the asynchronous data transfer. In this mode, the data error rate is very low but it requires at least 2 parallel channels to transmit the data, for that, the robot uses decoder and encoder modules (HT12D and HT12E).

Code snippet 1 : Data Transmission by the remote Controller to the robot

void transmit(short x)

```
//transmit the integer 'X'
{
        int i;
        j=x;
        set_data_enable();
        for(i=0;i<8;i++)       //send the data bit by bit
        {
                led_on();
                set_clock();
                if((j>>7)%2)
                        set_data();
                else
                        reset_data();
                send_data();
                delay(500);
                j<<=1;
                led_off();
                reset_clock();
                send_data();
                delay(500);
        }
        reset_data_enable();
        reset_data();
        reset_clock();
        send_data();
        delay(500);
        led_off();
}
```

Code snippet 2 : Data (destination station command) reception in the robot

```
if(P1IFG & BIT2)      //Wireless data reception
{
        if(start==0) return;
        short bit_count=0;
        recieved_data=0;
        while(P1IN & BIT2)
        {
                if(P1IN & BIT1)
                {
                        recieved_data<<=1;
                        if(P1IN & BIT0)
                                recieved_data+=1;
                        else
                                recieved_data+=0;
                        bit_count++;
                        while(P1IN & BIT1);
```

```
        }
    }
    if(bit_count!=8) recieved_data=0;
    if(recieved_data>0 && recieved_data<=station_count)
    {
        destination = recieved_data;
        start=0;
        print(recieved_data);
    }
    else
    {
        buz_on();
        delay(5000);
        buz_off();
    }
```

4.2.2. Motion Control Driver

The robotic movements are controlled by the Motion Control Driver, this driver sends the code to the motor driver IC through serial to parallel data transmission, corresponding to the action the robot wants to perform. The codes for forward, backward, right, left movements are described below :

Code snippet 3 : Motion-Control Driver

```
void forward()              //robot moves forward
{
    data[0]&=0;
    data[0]|=0x50;
    send_data();
}

 void backward()    -    data[0]|=0x28;
void left_turn()    -    data[0]|=0x10;
void right_turn()   -    data[0]|=0x40;
void rotate()       -    data[0]|=0x48;
void stop()         -    data[0]&=0;
```

4.2.3. Display Driver

The display driver is used to print the values on the screen. This is done by sending the corresponding LED-Glow code from the database of integer table. This table is stored in an array named as 'integers'. The value to be printed on the screen is first of all separated into single digits, then the corresponding LED-Glow code of each digit is sent from the table.

Code snippet 4 : Sending of LED-Glow value from the corresponding integer table

```
unsigned short integers[10]={0x02,0x4f,0x24,0x05,0x49,0x11,0x10,0x47,0x00,0x41};
void print(short x)
{
        data[1]&=BIT7;
        data[2]&=BIT7;
        data[1]|=integers[x%10];
        x/=10;
        data[2]|=integers[x%10];
        send_data();
}
```

4.2.4. Artificial Landmark Detection Driver

The Artificial Landmark Detection Driver walks on the landmark and sends the corresponding value on the arrival of station or a junction. This driver helps the robot to know if its standing on a junction or a station. The robot uses only this driver to scan the network and build the corresponding tree-map of it.

Code snippet 5 : Artificial Landmark Detection Driver

```
short move_on_line()
{
        short sensors,i;
        while(1)
        {
                sensors = P2IN;
                sensors = sensors & (r+m+l);
                sensors>>=3;
                switch(sensors)
                {
                case 0 :        //on station
                                        stop();
                                        delay(2000);
                                        for(i=0;i<20;i++)
```

```
                                {
                                        backward();
                                        delay(1);
                                        stop();
                                        delay(8);
                                }
                                return 0;
                        case 4 :        //line on left
                        case 6 :
                                        right_turn();
                                        delay(10);
                                        stop();
                                        delay(20);
                                        break;
                        default :
                        case 2 :        //line at centre
                                        forward();
                                        delay(10);
                                        stop();
                                        delay(20);
                                        break;
                        case 1 :        //line on right
                        case 3 :        left_turn();
                                        delay(10);
                                        stop();
                                        delay(20);
                                        break;
                        case 7 :
                                        for(i=0;i<2;i++)
                                        {
                                                forward();
                                                delay(5);
                                                stop();
                                                delay(15);
                                        }
                                        stop();
                                        return 1;
                        }
                        stop();
                        delay(10);
        }
}
```

4.2.5. Collision Avoidance Driver

The Collision Avoidance Driver is a simple driver which pauses the execution of the system on detection of any object in front of the robot.

Code snippet 6 : Collision Avoidance Driver

```
stop();
while(P2IN & 0)
{
        buz_on();
        delay(500);
        buz_off();
        delay(1000);
}
```

4.2.6. Other Small Drivers

ROS has other small drivers like Buzzer Controller, Button Controller and Landmark indication Driver. Buzzer Controller Driver controls the beep sounds made for indication of arrival of a station and for the indication of an obstacle in front of the robot. Button Controller Driver controls the working of the buttons. There two buttons are helpful in providing vital information to the robot i.e. when to start the scanning (Button 1) and when to reset the station to station 1 (Button 2). The Landmark Indication Driver indicates the detection of the landmark using 3 LEDs mounted on the robot. This helps the user and the designer to calibrate the sensors with the landmarks.

4.3. Flow Chart of the Robotic Operating System (ROS)

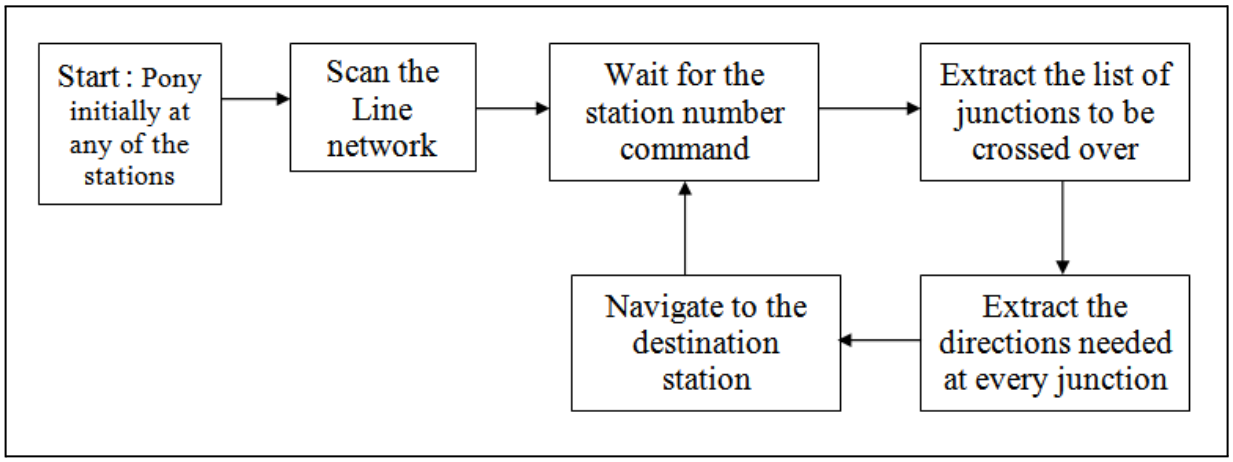

Figure 4.2: Flow Chart of working of ROS