# Final Report

## Title of the Project

Expression planter for determination of temperature, moisture and light conditions for effective growth of plants.


**Team Members**

Ambati Somnath – 21BCE9501

Paipalli Sai Sathwik – 21BCE9340

Devapatla Reddy Nithish kumar – 21BCE9054

Mudimi Charan - 21BEC7211

Avula Netraditya - 21BCE7795

Reddi Vinay Kumar - 21BCE9157

**Guided By**

Dr. K Venkateswarlu

**Vellore Institute of Technology**

VIT-AP University

G-30, Inavolu

Beside AP Secretariat Amaravati

Andhra Pradesh

522237

## Abstract:

The expression planter project is a creative and innovative approach to combining art, nature, and technology. The Expression planter is a modern, intelligent planter designed to provide the ideal environment for plants to grow. The project involves designing and building planters that can display expressions through different means, such as light, Moisture and Temperature. These planters can be customized to suit various environments, from indoor spaces to outdoor gardens. The expression planter project offers a unique way to incorporate living art into homes and public spaces while encouraging people to connect with nature in a playful and interactive way. The project involves the creation of a smart planter that allows users to express themselves through digital images and designs that are displayed on the planter's surface. The planter uses a microcontroller and sensors to monitor the plant's health and environmental conditions, such as moisture levels, temperature, and light. Users can interact with the planter through a mobile application that allows them to see the values in the graph with the history. The project offers an affordable and accessible solution to create a sustainable and aesthetic indoor gardening environment, suitable for home or office decoration, and a healthy lifestyle. With the Expression Planter, users can enjoy the benefits of indoor gardening while also promoting sustainability and healthy living. The project is an innovative solution that combines technology and nature to create a fun and interactive way to grow plants in small spaces. The project is intended to promote sustainable urban agriculture and enable users to grow healthy plants with minimal effort and also provides a fun and educational way for people to explore electronics and programming while promoting a deeper appreciation for the natural world.

The Expression planter displays 6 expressions which are directly related to our plant's needs-

**Thirsty**: Displays thirsty whenever the soil moisture is too low for the plant and is looking for some water.

**Hot**: Displayed whenever it is too hot.

**Freeze**: When the temperature drops too low, plant needs some warmth.

**Savory**: When the plant gets some water, it's too delicious.

**Happy**: When everything is perfect, it is too happy.

# Index

## Introduction

The Expression Planter is a fascinating and innovative project that merges the worlds of technology and nature. It is an interactive planter that is capable of expressing emotions using light and sound, allowing users to communicate with their plants in a unique and engaging way.

The project is built on the Arduino, which is an open-source electronics platform that provides a flexible and easy-to-use environment for creating interactive projects. The planter includes various sensors that monitor environmental factors like temperature, humidity, and soil moisture, which are used to adjust the planter's behaviour and express emotions based on the plant's current state.

The Expression Planter is an excellent project that combines technology and horticulture in a unique and creative way. It provides a fun and educational opportunity for people to explore the possibilities of electronics and programming while also promoting a deeper appreciation for the natural world.

## Background

The Expression Planter is a relatively new project that combines the power of technology with the beauty of nature. This project is built on the Arduino platform, which is an open-source electronics platform that provides an easy-to-use environment for creating interactive projects. The Expression Planter project takes advantage of this platform to create a unique and interactive planter that incorporates various sensors and electronic components to monitor and tells the user about the planter's behaviour based on the plant's needs.

The Expression Planter project takes this concept to the next level by incorporating features that allow the planter to express emotions using LFT display module. This interactive feature allows users to communicate with their plants in a unique and engaging way, creating a deeper connection between the user and the plant.

## Problem Definition

Taking care of plants are more important. Missing even a small scheduled task for the plant can result in the drying up of the plant and even its death. We don't know when the plants need Sunlight and water for it, even excess of it leads to plants death. This is a big problem in houses, where not shifting the plant to place of the sunlight at the required time. Not watering the plant by checking the amount of moisture in the soil of the pot. All these processes or tasks can result in that the plant in unhealthy plants or even plant death. There is a need for a solution that makes plant care simple and easy, providing personalized care instructions based on each plant's requirements.

## Objectives

- The objectives of this project is to design and build a functional prototype of the smart expression planter using Arduino technology and developing a user-friendly interface for controlling and displaying the emotions according to the plants requirements, through the moisture, light, and temperature sensors.

- Incorporate sensors to measure and analyze data such as soil moisture, humidity, and temperature.

- Develop an algorithm to display expressions of the plant care routines based on sensor data, ensuring optimal growth conditions.

- Integrate a customizable display system to showcase messages, emojis, or even art or gifs even.

- To create a low-cost and environmentally sustainable Expression Planter that can be easily assembled and maintained by individuals and communities.

- To develop an open-source project that can be easily modified and adapted to meet the specific needs and requirements of different users and applications.

## Methodology/Procedure

- We gathered the required materials listed in Table 1.

**Table-1**

| S.no. | required materials |
|---|---|
| 1 | Arduino UNO |
| 2 | Moisture sensor |
| 3 | Temperature sensor |
| 4 | Light Sensor |
| 5 | TFT Display Module |
| 6 | Amplifier Module |
| 7 | Wi-fi module (ESP8266) |
| 8 | Perforated board |
| 9 | Silicon wires |
| 10 | USB power cable |

- Connected the LCD display module, Temperature sensor, Moisture sensor and Light sensor to the Arduino by wires with perforated board.
- Then placed Arduino in the front of the Pot and attach the Display module on the Arduino by covering it.
- Then we secured the LDR module and the temperature sensor at beside each other where, the both sensors are visible to the outside.
- Attached the ADC Amplifier module and Arduino by means of the strong double-sided tape.
- After we connected the USB module with the Arduino and glued the base and the outer cover.
- Then glue all the components including Perforated board which is attached with Wi-fi module.
- Then we inserted the moisture sensor into the soil where the plant and soil is already set and made remaining connections.
- Finally, powered the Arduino with USB cable through laptop and then the planter displays emotions as required.

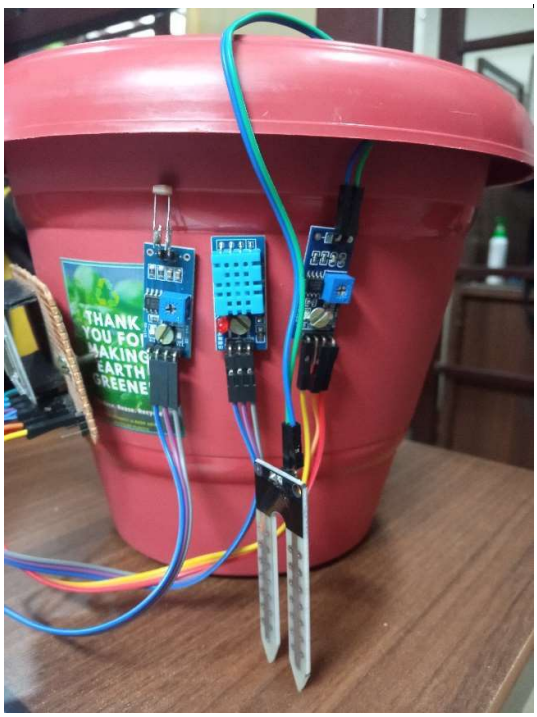


**Fig. 1 (Light, Temperature and Moisture sensor)**

**Fig. 2 (Wi-fi module ESP8266)**

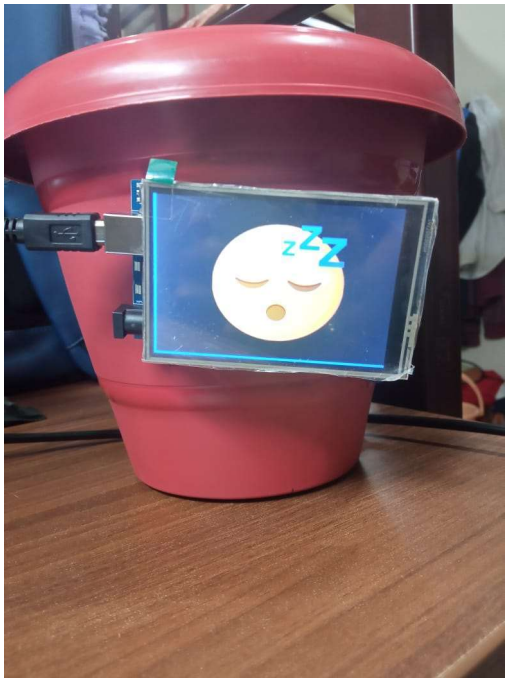
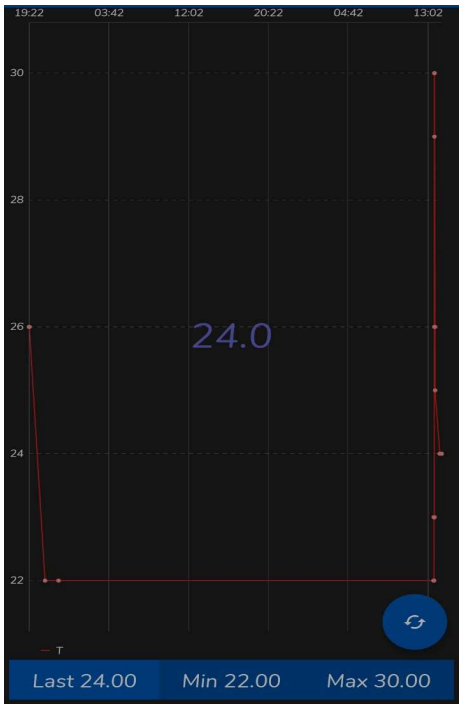


**Fig. 3 (Displaying Sleepy Emoji).**

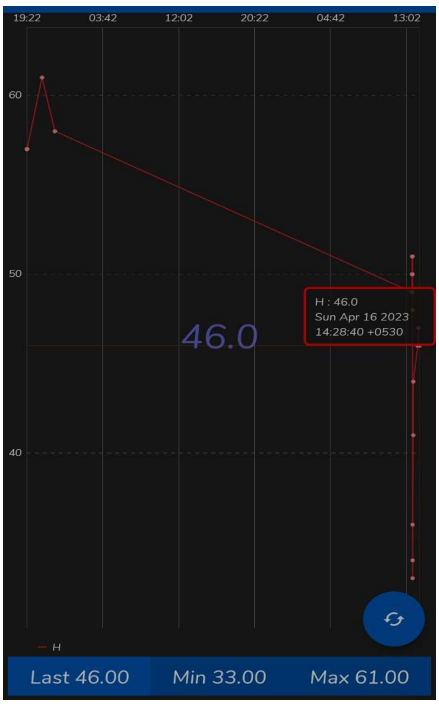
**Fig. 4(Displaying Happy Emoji).**

## Results and Discussion

The results of the Expression Planter show that it is a viable solution to take care of plants growth. The Expression Planter consisted of a pot with a soil moisture sensor, a temperature and humidity sensor, a light sensor, and a TFT display module that showed the real-time data and allowed users to take care the plant in the required time. Expression Planter demonstrated several benefits and challenges in terms of sustainability, accessibility, and innovation. The Expression Planter was able to regulate the plant growth and maintain the soil moisture, temperature, and light conditions within the optimal range, which resulted in healthier and more vibrant plants.




Readings of the Temperature values in Graph varying in different situations.




Reading the the Humidity or Moisture levels in Graph varying in different situations.

## Conclusion and Future Scope

The Expression Planter has shown great potential as a sustainable solution for enhancing plant growth and promoting food security. The Smart Expression Planter has successfully regulated the soil moisture, temperature, and light conditions within the optimal range, resulting in healthier and more vibrant plants. This technology has the potential to revolutionize the way plants are grown, making users easier and more efficient to grow healthy plants in any environment.

The Expression Planter has several potential areas of improvement and expansion for future research and development. One of the possible directions is to enhance the Expression Planter's automation and remote-control features by integrating machine learning and artificial intelligence algorithms that can predict and adjust the plant growth conditions based on the historical data and user preferences. Another potential direction is to explore the Expression Planter's integration with other smart home and IoT devices, such as voice assistants and smart lighting systems, to create a more seamless and efficient user experience. Moreover, the Expression Planter can be customized to support various plant species and environmental conditions, which can expand its application and impact in different regions and communities.

## References


https://www.etechnophiles.com/raspberry-pi-pico-pinout-specifications-datasheet-in-detail/#:~:text=Raspberry%20Pi%20Pico%20is%20made,USB%20port%2C%20and%20power%20pin

https://wiki.dfrobot.com/Capacitive_Soil_Moisture_Sensor_SKU_SEN0193

https://circuitdigest.com/microcontroller-projects/interfacing-lm35-sensor-with-arduino

https://www.sunrom.com/p/light-sensing-module-ldr

https://www.hackster.io/coderscafe/raspberry-pi-4-headless-817c01

https://youtu.be/aVCWLk10sAw

https://youtu.be/aVCWLk10sAw

https://www.thisiswhyimbroke.com/lua-smart-planter/

https://www.instructables.com/Fyt%C3%B3-Turn-Your-Plant-Into-Pet/

https://www.youtube.com/watch?v=6xfGRa70Xao

## Codes in Appendix

```
#include <SPI.h>
//#define USE_SDFAT
#include <SD.h>
#include <Adafruit_GFX.h>
#include <MCUFRIEND_kbv.h>
MCUFRIEND_kbv tft;
#include <TouchScreen.h>
const int XP=8,XM=A2,YP=A3,YM=9;
const int TS_LEFT=180,TS_RT=805,TS_TOP=935,TS_BOT=130;
TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);
TSPoint tp;
int buz=A5;

int scr=1;
unsigned char itt[8];
#define MINPRESSURE 200
#define MAXPRESSURE 1000
int sub=0,x;
char cnfm=0;
#if defined(ESP32)
#define SD_CS     5
#else
#define SD_CS     10
#endif
#define NAMEMATCH ""        // "" matches any name
//#define NAMEMATCH "tiger"   // tiger.bmp
```

```
#define PALETTEDEPTH   0     // do not support Palette modes
//#define PALETTEDEPTH   8     // support 256-colour Palette

char namebuf[6] = "/";   //BMP files in root directory
//char namebuf[32] = "/bitmaps/";  //BMP directory e.g. files in /bitmaps/*.bmp
File root;
int pathlen,prv=0;
unsigned char it11,it21,it31,it41,cnt=0,it12,it22,it32,it42,canc=0;
int tot,cnt1=0,cnt2=0;
void setup()
{
   uint16_t ID;
   Serial.begin(9600);
   ID = tft.readID();
   pinMode(buz,OUTPUT);
   digitalWrite(buz,1);
   delay(300);
   digitalWrite(buz,0);
   if (ID == 0x0D3D3) ID = 0x9481;
   tft.begin(ID);
   tft.fillScreen(0x001F);
   tft.setTextColor(0xFFFF, 0x0000);
   bool good = SD.begin(SD_CS);
   if (!good) {
      while (1);
   }
   root = SD.open(namebuf);
   pathlen = strlen(namebuf);
   delay(1000);
   int x = showBMP("/welcome.bmp", 5, 5);
}
void loop()
```

```
{
 if(Serial.available())
 {
  int x=Serial.read();
  Serial.write(x);
  if(x!=prv)
  {
  if(x=='4')
  {
   Serial.write("Happy");
   prv=x;
   int x = showBMP("/happy.bmp", 5, 5);
  }
     if(x=='2')
  {
   Serial.write("Hot");
   prv=x;
   int x = showBMP("/hot.bmp", 5, 5);
  }
     if(x=='3')
  {
   Serial.write("Dark");
   prv=x;
   int x = showBMP("/1.bmp", 5, 5);
  }
  if(x=='1')
  {
   Serial.write("Thrust");
   prv=x;
   int x = showBMP("/1.bmp", 5, 5);
  }
  }
```

```
  }
}
#define BMPIMAGEOFFSET 54
#define BUFFPIXEL      20
uint16_t read16(File& f) {
    uint16_t result;         // read little-endian
    f.read((uint8_t*)&result, sizeof(result));
    return result;
}
uint32_t read32(File& f) {
    uint32_t result;
    f.read((uint8_t*)&result, sizeof(result));
    return result;
}
uint8_t showBMP(char *nm, int x, int y)
{
    File bmpFile;
    int bmpWidth, bmpHeight;    // W+H in pixels
    uint8_t bmpDepth;           // Bit depth (currently must be 24, 16, 8, 4, 1)
    uint32_t bmpImageoffset;    // Start of image data in file
    uint32_t rowSize;           // Not always = bmpWidth; may have padding
    uint8_t sdbuffer[3 * BUFFPIXEL];    // pixel in buffer (R+G+B per pixel)
    uint16_t lcdbuffer[(1 << PALETTEDEPTH) + BUFFPIXEL], *palette = NULL;
    uint8_t bitmask, bitshift;
    boolean flip = true;        // BMP is stored bottom-to-top
    int w, h, row, col, lcdbufsiz = (1 << PALETTEDEPTH) + BUFFPIXEL, buffidx;
    uint32_t pos;               // seek position
    boolean is565 = false;      //

    uint16_t bmpID;
    uint16_t n;                 // blocks read
    uint8_t ret
```

```
if ((x >= tft.width()) || (y >= tft.height()))
   return 1;               // off screen
bmpFile = SD.open(nm);      // Parse BMP header
bmpID = read16(bmpFile);    // BMP signature
(void) read32(bmpFile);     // Read & ignore file size
(void) read32(bmpFile);     // Read & ignore creator bytes
bmpImageoffset = read32(bmpFile);       // Start of image data
(void) read32(bmpFile);     // Read & ignore DIB header size
bmpWidth = read32(bmpFile);
bmpHeight = read32(bmpFile);
n = read16(bmpFile);        // # planes -- must be '1'
bmpDepth = read16(bmpFile); // bits per pixel
pos = read32(bmpFile);      // format
if (bmpID != 0x4D42) ret = 2; // bad ID
else if (n != 1) ret = 3;   // too many planes
else if (pos != 0 && pos != 3) ret = 4; // format: 0 = uncompressed, 3 = 565
else if (bmpDepth < 16 && bmpDepth > PALETTEDEPTH) ret = 5; // palette
else {
   bool first = true;
   is565 = (pos == 3);               // ?already in 16-bit format
   // BMP rows are padded (if needed) to 4-byte boundary
   rowSize = (bmpWidth * bmpDepth / 8 + 3) & ~3;
   if (bmpHeight < 0) {              // If negative, image is in top-down order.
      bmpHeight = -bmpHeight;
      flip = false;
   }
   w = bmpWidth;
   h = bmpHeight;
   if ((x + w) >= tft.width())       // Crop area to be loaded
      w = tft.width() - x;
   if ((y + h) >= tft.height())      //
      h = tft.height() - y;
```

```
    if (bmpDepth <= PALETTEDEPTH) {   // these modes have separate palette
      //bmpFile.seek(BMPIMAGEOFFSET); //palette is always @ 54
      bmpFile.seek(bmpImageoffset - (4<<bmpDepth)); //54 for regular, diff for colorsimportant
      bitmask = 0xFF;
      if (bmpDepth < 8)
        bitmask >>= bmpDepth;
      bitshift = 8 - bmpDepth;
      n = 1 << bmpDepth;
      lcdbufsiz -= n;
      palette = lcdbuffer + lcdbufsiz;
      for (col = 0; col < n; col++) {
        pos = read32(bmpFile);    //map palette to 5-6-5
        palette[col] = ((pos & 0x0000F8) >> 3) | ((pos & 0x00FC00) >> 5) | ((pos & 0xF80000) >> 8);
      }
    }
    // Set TFT address window to clipped image bounds
    tft.setAddrWindow(x, y, x + w - 1, y + h - 1);
    for (row = 0; row < h; row++) { // For each scanline...
      // Seek to start of scan line.  It might seem labor-
      // intensive to be doing this on every line, but this
      // method covers a lot of gritty details like cropping
      // and scanline padding.  Also, the seek only takes
      // place if the file position actually needs to change
      // (avoids a lot of cluster math in SD library).
      uint8_t r, g, b, *sdptr;
      int lcdidx, lcdleft;
      if (flip)   // Bitmap is stored bottom-to-top order (normal BMP)
        pos = bmpImageoffset + (bmpHeight - 1 - row) * rowSize;
      else        // Bitmap is stored top-to-bottom
        pos = bmpImageoffset + row * rowSize;
      if (bmpFile.position() != pos) { // Need seek?
```

```
    bmpFile.seek(pos);
    buffidx = sizeof(sdbuffer); // Force buffer reload
  }
  for (col = 0; col < w; ) {  //pixels in row
    lcdleft = w - col;
    if (lcdleft > lcdbufsiz) lcdleft = lcdbufsiz;
    for (lcdidx = 0; lcdidx < lcdleft; lcdidx++) { // buffer at a time
      uint16_t color;
      // Time to read more pixel data?
      if (buffidx >= sizeof(sdbuffer)) { // Indeed
        bmpFile.read(sdbuffer, sizeof(sdbuffer));
        buffidx = 0; // Set index to beginning
        r = 0;
      }
      switch (bmpDepth) {          // Convert pixel from BMP to TFT format
        case 24:
          b = sdbuffer[buffidx++];
          g = sdbuffer[buffidx++];
          r = sdbuffer[buffidx++];
          color = tft.color565(r, g, b);
          break;
        case 16:
          b = sdbuffer[buffidx++];
          r = sdbuffer[buffidx++];
          if (is565)
            color = (r << 8) | (b);
          else
            color = (r << 9) | ((b & 0xE0) << 1) | (b & 0x1F);
          break;
        case 1:
        case 4:
        case 8:
```

```
                    if (r == 0)
                        b = sdbuffer[buffidx++], r = 8;
                    color = palette[(b >> bitshift) & bitmask];
                    r -= bmpDepth;
                    b <<= bmpDepth;
                    break;
                }
                lcdbuffer[lcdidx] = color;

            }
            tft.pushColors(lcdbuffer, lcdidx, first);
            first = false;
            col += lcdidx;
        }           // end cols
    }               // end rows
    tft.setAddrWindow(0, 0, tft.width() - 1, tft.height() - 1); //restore full screen
    ret = 0;        // good render
  }
  bmpFile.close();
  return (ret);
}
```