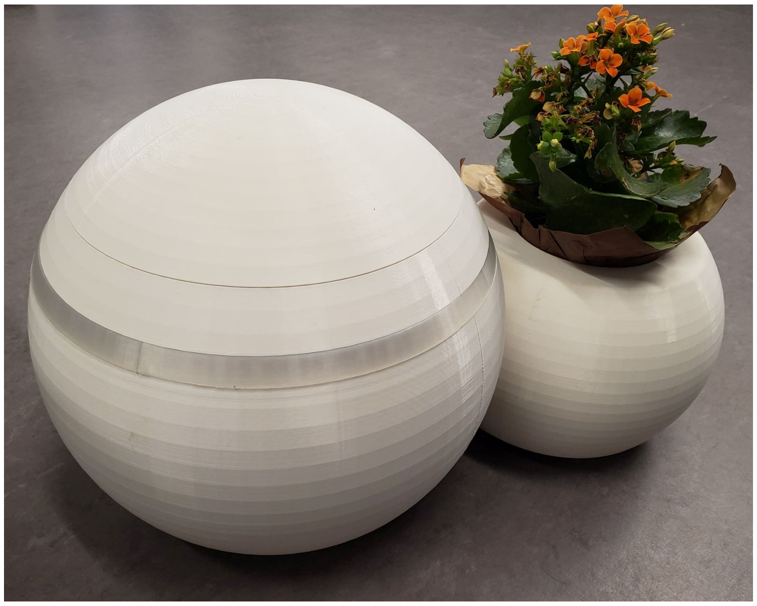
growbot

Christine Yuan (cjy26)
Cuyler Crandall (csc254)
Deanna Kocher (drk225)

DEA 6210
December 17, 2019

Contents



1 Abstract

In this project we present Growbot, the half of the Happy Plant Happy Child system comprised of the Growbot and the Growall. The system teaches children positive values, such as responsibility, patience, and care, by encouraging and guiding them to take care of a plant. The child primarily interacts with the Growbot, which tracks the health of a specific plant. When the plant has particular needs, such as water or light, the Growbot nudges the child to complete these actions. If the child successfully takes care of the plant over a long period of time, the Growall will being more intricate to positively reinforce the child's actions.

2 Scenario

Elle is a 5 year old girl who is exploring new interests. Her parents want to instill good values in her at an early age such as responsibility, patience, and care. They install a plant system in her room to teach her how to take care of something living and hope that she will learn other positive lessons as well. A small robot serves as the teaching mechanism to guide Elle to take care of her plant. The robot has a lighting system which Elle must use to provide light to her plant. She is responsible for changing the lightbulb whenever necessary. The robot will also tug the plant around to remind Elle to take care of it. As Elle successfully takes care of the plant, her room will change to positively reinforce Elle's actions. Over time, flower pieces will shift as if they are growing along her walls. The room will seem alive to reflect the plant that Elle is taking care of.

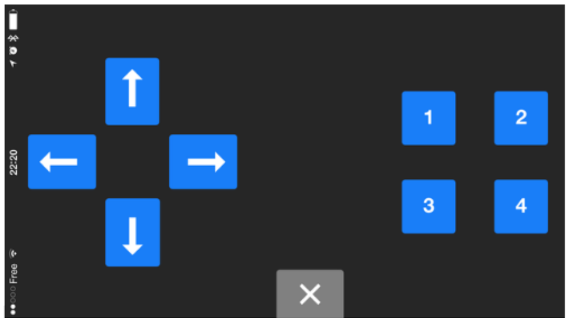
3 Operation

The Growbot prototype is a differential drive mobile robot that in controlled by the Adafruit Bluetooth shield controller. It has eight controls, five of which relate to drive steering. The controls all relate to the "plant state" and help instruct the child on the best way to care for the plant. For this first prototype, the robot-plant connection is done through "wizard-of-oz" techniques, and the robot is controlled entirely by Bluetooth.

There are three robot behaviors related to the plant state. If the plant needs water, the robot LED strip will pulse blue rapidly. If the plant needs light (sunlight or artificial Growbot light), the LED strip pulses yellow in the same way. When the child obliges the plant by giving it what it needs, the LED strip pulses green at a slower rate that is similar to breathing.

The robot is free to move around on its own with the driving commands: forward, backward, turn in place to the right, turn in place to the left. Magnets above the light switch and power adaptor allow the robot to magnetically connect to the plant, which has multiple magnets to allow for easy connection from different orientation. Whenever the plant needs something like water or sunlight, the robot can link to the plant and push it over to the child. In this way, even if the child is occupied and would otherwise forget about the plant, the robot can bring it to their attention. To disconnect from the plant, the robot can turn in place and overcome the magnet shear strength.

3.1 Bluetooth Controls



Button	Action
$\Uparrow$	Drive Forward, full speed
$\Downarrow$	Drive Backward, full speed
$\Leftarrow$	Turn in place, counterclockwise
$\Rightarrow$	Turn in place, clockwise
1	Pulse blue – the plant needs water
2	Pulse yellow – the planet needs light
3	Pulse green – the plant is properly cared for
4	Stop driving commend – sets motor speeds to 0

Figure 1: Robot Control Scheme in the Bluetooth App

4 Construction

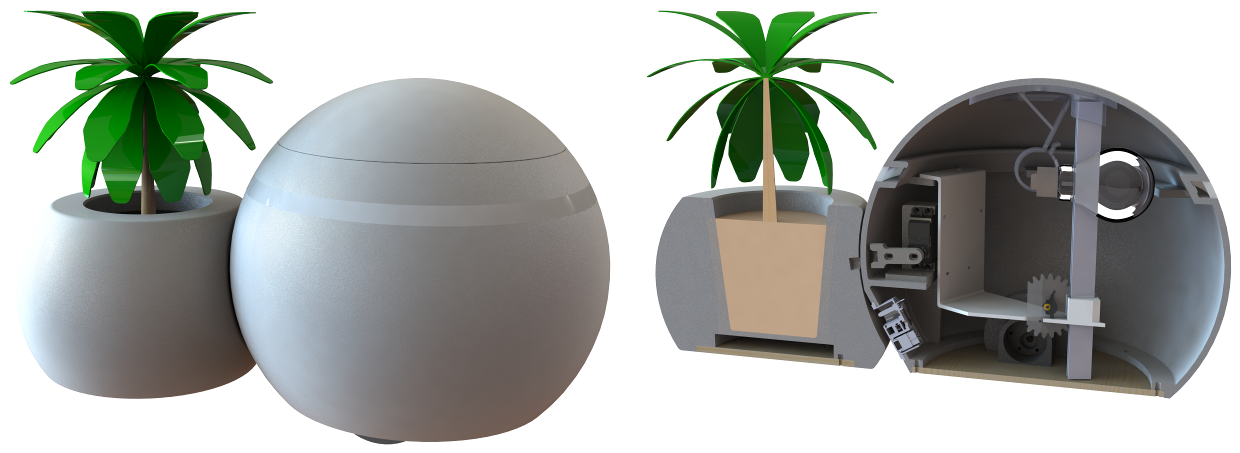

(a) Full Design (b) Cross Section

Figure 2: CAD of Growbot

At a system level, the Growbot was constructed based on the system diagram presented in Figure 3, with additional 3D printed and laser cut structures holding everything together. The subsystems of Growbot can be roughly broken down into the Shell, Drivebase, Electronics Bay/Lid, and the Follow Bot planter.

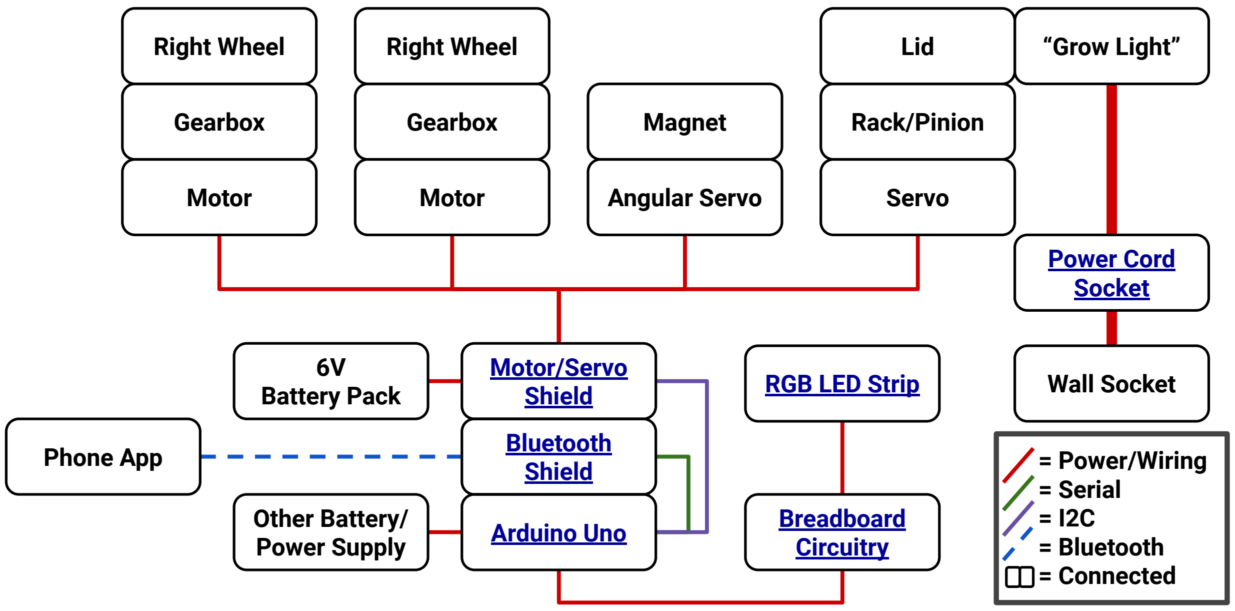


Figure 3: System Diagram of the Growbot

4.1 Shell

The Growbot is primarily constructed of a large 3D printed shell which provides mounting points for a drive base, electronics bay, and lid system. Also baked into the shell's design is a RGB LED strip with a translucent cover and a computer power cord adapter used to power the lightbulb via a wall socket. The drive base and electronic bay mount to the shell using #4-40 screws and heat-set insets in rings of holes at the top and bottom of the robot.

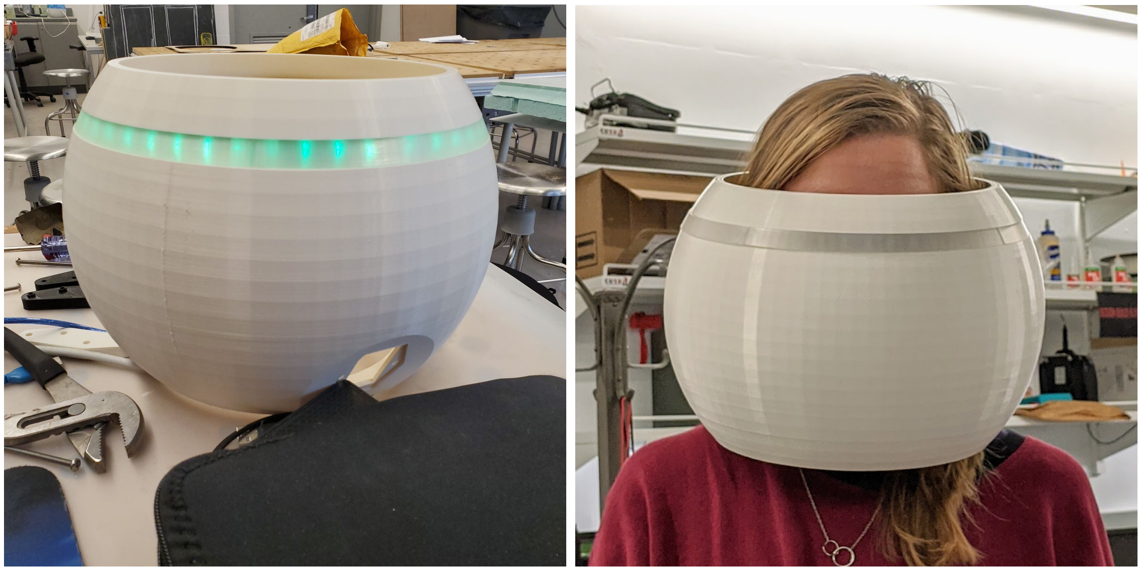

(a) Test of LED Strip (b) Real Life Scale of Shell

Figure 4: Construction of the Shell

4.2 Drivebase

The removable drivebase on the Growbot allows for the entire bottom of the robot to be removed, allowing for easier access to internal features, such as the magnet servo's mounting and the power adapter's nuts. The drivebase itself is a laser cut wooden disk, and mounted to it are a ball caster and two $\sim$ 60rpm gearmotors which power the Growbot's motion (at a brisk $1\,\mathrm{ft\,s^{-1}}$ pace). The mounts for the gearmotors and wheels attached to them are 3D printed, with rubber bands added to the wheels for traction.

4.3 Electronics Bay/Lid

The electronics bay inside the Growbot is a 3D printed drop-in frame which houses mounting for the Adruino brain of the robot, guides for the light mount/lid support, and the servo which drives the lid's rack and pinion actuation. The lid support which the electronics bay servo raises and lowers has a "grow light" attached to it by a 3D printed clip. This light is turned on or off by a switch on the computer power cable adapter port on the rear side of the robot.

4.4 Follow Bot (Planter)

Plants the Growbot is caring for are pulled along behind it in a spherical planter riding on three ball casters. The planter and Growbot are coupled together by a magnetic connection between

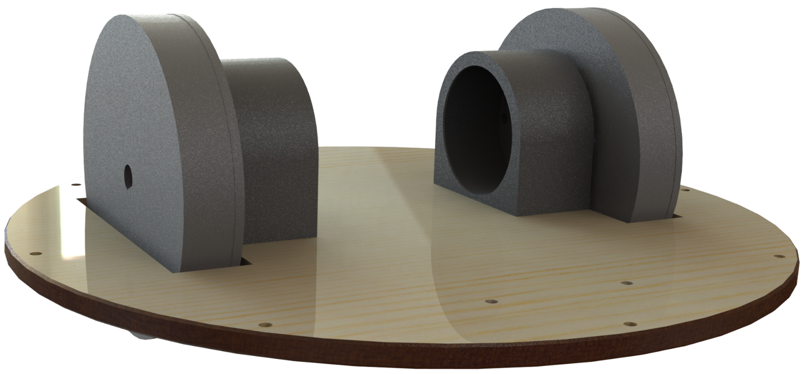
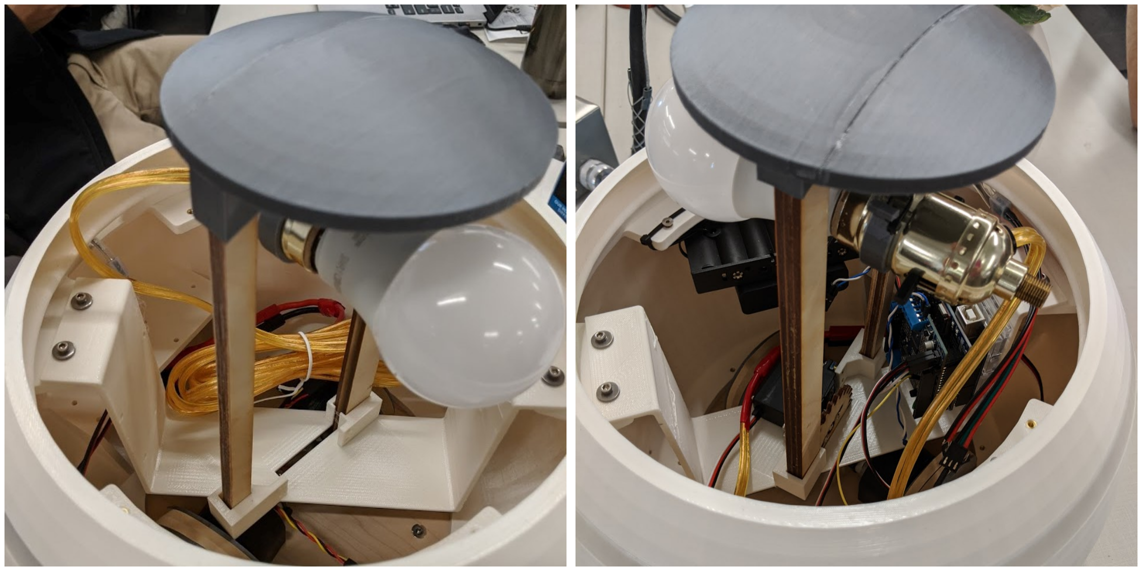


Figure 5: Drivebase CAD Model (Manufactured Parts)



(a) without Electronics (b) with Electronics and Servos

Figure 6: The Electronics Bay Inside the Growbot

magnets mounted to a servo within the Growbot and in three press-fit recesses on the planter. A curved indent matching the curvature of the Growbot helps to align the planter to the Growbot when they are to be attached together since the growbot can drive into the recess, then spin until the magnets are properly aligned. The planter is constructed of a 3D printed body, with its drive base constructed similarly to the Growbot's: a laser cut sheet of plywood, attached to the printed body using #4-40 heat set inserts.

5 Discussion

The Growbot was capable of achieving most of the desired "feel" of the robot, but ended up lacking in a number of functional areas. The Bluetooth control app was sluggish and our program was prone to getting stuck believing a button to be pressed when it was not, issues which resulted in imprecise control and many takes required for filming as the robot would rarely perform as intended. Additionally, the rack and pinion mechanism intended to raise and lower the lid bound up, meaning

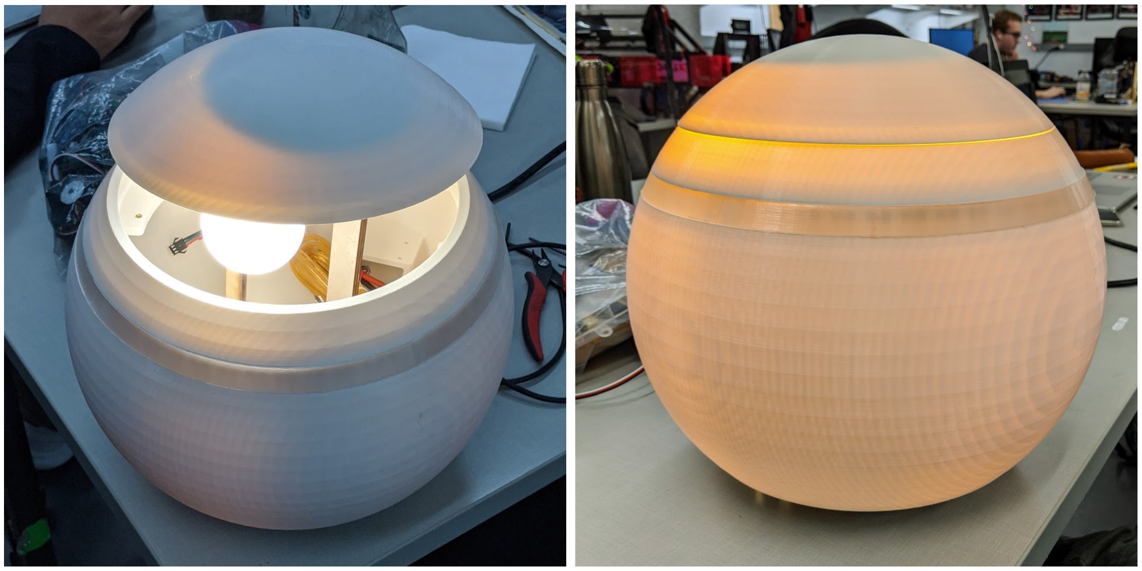

(a) when Open (b) when Closed "Nightlight Mode"

Figure 7: Effects of the Grow Light

that active control of the lid was lost. For filming purposes we resorted to a cut between the robot getting into position and us raising the lid and wedging it in that position. Both the magnet servo and lid raising/lowering servo were never fully implemented, the lid servo because the mechanism it drove was not functional, and the magnet servo because keeping the magnets in a static position and spinning the Growbot to attach/detach was equally effective. Finally, the Growbot did not have the required mass to reliably *pull* the loaded planter across surfaces which weren't perfectly flat, meaning that we had to resort to pushing the plant around for filming. Though this was a fine solution for a single plant in straight lines, it poses challenges for the future vision of complex paths and possibly pulling a train of plants.

Despite all of the above issues, we were able to successfully achieve more than enough to get the Growbot functional for filming and demonstrating the concepts behind the system. The Growbot's adorable trundling around was popular with our test child, the diffused LED signals gave clear indication of the plant's needs, and the grow light provided warm illumination for the plants. With some refinement the Growbot could easily become fully functional in all the ways we initially intended it to be.

6 Future Work

Moving forward, the robot needs to be programmed to respond to a plant's needs. In this situation, the robot would not need to be Bluetooth controlled. For this condition, we would need a number of different sensors, as shown in the table below.

Component	**Qty.**
Soil Moisture Sensor	1
Sunlight/UV Sensor	1
Growth Sensor (Mass)	1
Grow Light	1

To measure plant growth, we chose to use a mass sensor so that growth could be gauged regardless of the direction the plant grows (i.e. a shrub vs a sunflower). In this way, as long as the plant is generally getting bigger, it will be on average reflected by the mass of the entire pot. This growth metric serves primarily to alert the Growall (discussed later) of the child and plant's long term progress.

In order to implement these sensors with the plant, we would be adding an additional Arduino to the plant carrier, and also adding a transmitter and receiver for communication. With this communication, the Growbot itself would be much more autonomous: able to receive signals and react with the LED strip or drive to the robot (although this would require a way to track the location of both robots).

Switching from the Bluetooth control will also make the drive mechanism of the robot more functional. In its current state, the robot can only push the plant in a straight line direction because the turning arrows turn the robot in place and disconnect the robot from the plant. With more magnets and more comprehensive steering, Growbot would be able to push the plant across a flat surface along a path, rather than just in a straight line.

Further, the lid raising mechanism for Growbot needs to be redesigned. The rack and pinion setup that was originally designed binds up due to tolerancing. For a future revision we imagine that the lid would not lift straight up with a rack and pinion, but would hinge from one side. Lastly, in its current state, the light is powered through a separate cord that has to be plugged in to the robot and then plugged into an outlet. For future revisions, it would be better to have a cord that spools within the robot and can auto-retract when not in use, or if the light was powered wirelessly. Either of these options would make the Growbot more useable for a child.

A Appendix

A.1 Related Links

A video of the Growbot in action can be found on YouTube here.

A.2 Bill of Materials

Table 1: Purchased Components

Component	Unit	Qty.
LED Strip	ft.	3
Motors	ea.	2
5/8" Casters	ea.	5
12V Battery Carrier	ea.	1
Adafruit Bluetooth Shield	ea.	1
Adafruit Motor/Servo Shield	ea.	1
1/8" Plywood Sheet	ft^2	1
3/16" Plywood Sheet	ft^2	1
Zip Ties	ea.	$\sim$ 8
Male-Male Wire Connectors	ea.	$\sim$ 15
#4-40 Heat Set Inserts	ea.	$\sim$ 20
5/16" Screws, assorted	ea.	$\sim$ 5
#4-40 Screws, assorted	ea.	$\sim$ 20

Table 2: Fabricated Components

Component	Qty.	Manufacturing Method
Robot Body	1	3D Printed
Robot Hat	1	3D Printed
Bulb Mount	1	3D Printed
Plant Carrier	1	3D Printed
Wheels	2	3D Printed
Motor Mount	2	3D Printed
Electronics Bay	1	3D Printed
Magnet Servo Mount	1	3D Printed
Magnet Servo Horn	1	3D Printed
LED Strip Cover Quadrent	4	3D Printed (Translucent filament)
Wheel Bases	2	Laser Cut
Rack Guide	1	Laser Cut
Rack	1	Laser Cut
Pinion Gear	1	Laser Cut

A.3 Arduino Code

Libraries that must be imported (all available for download in the Arduino interface):

- Adafruit BluefruitLE nRF51
- Adafruit Motorshield v2 Library
- FastLED
- Servo

Accessible files (in the Arduino folder)

- BluefruitConfig.h
- packetParser.cpp

```
/*********************************************************************
 This is an example for our nRF51822 based Bluefruit LE modules

 Pick one up today in the adafruit shop!

 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!

 MIT license, check LICENSE for more information
 All text above, and the splash screen below must be included in
 any redistribution
*********************************************************************/

#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"
#include <FastLED.h>
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <Servo.h>


#include "BluefruitConfig.h"

#if SOFTWARE_SERIAL_AVAILABLE
  #include <SoftwareSerial.h>
#endif

```

```
/*=========================================================================
    APPLICATION SETTINGS

        FACTORYRESET_ENABLE        Perform a factory reset when running this sketch
   
                                    Enabling this will put your Bluefruit LE module
                                in a 'known good' state and clear any config
                                data set in previous sketches or projects, so
                                    running this at least once is a good idea.
   
                                    When deploying your project, however, you will
                                want to disable factory reset by setting this
                                value to 0.    If you are making changes to your
                                    Bluefruit LE device via AT commands, and those
                                changes aren't persisting across resets, this
                                is the reason why.    Factory reset will erase
                                the non-volatile memory where config data is
                                stored, setting it back to factory default
                                values.
       
                                    Some sketches that require you to bond to a
                                central device (HID mouse, keyboard, etc.)
                                won't work at all with this feature enabled
                                since the factory reset will clear all of the
                                bonding data stored on the chip, meaning the
                                central device won't be able to reconnect.
    MINIMUM_FIRMWARE_VERSION  Minimum firmware version to have some new
```

```
        features
    MODE_LED_BEHAVIOUR        LED activity, valid options are
                              "DISABLE" or "MODE" or "BLEUART" or
                              "HWUART"  or "SPI"  or "MANUAL"
    -----------------------------------------------------------------------
    */
    #define FACTORYRESET_ENABLE         1
    #define MINIMUM_FIRMWARE_VERSION    "0.6.6"
    #define MODE_LED_BEHAVIOUR          "MODE"

    //LED Strip Pins
    #define NUM_LEDS 47
    #define DATA_PIN 3
/*=========================================================================*/

// Create the bluefruit object, either software serial...uncomment these lines
/*
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_PIN);

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
*/

/* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line */
// Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);

/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/IRQ/RST */
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

/* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user selected CS/IRQ/RST */
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
//                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
//                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);


// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  while (1);
}

// function prototypes over in packetparser.cpp
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout);
float parsefloat(uint8_t *buffer);
void printHex(const uint8_t * data, const uint32_t numBytes);

// the packet buffer
extern uint8_t packetbuffer[];

/*
    **************************************************************************
    */
/*!
 *  external components we are adding for growbot
 *  LED Strip
 *  Motors
 *  Servos
 */
/*
    ***************************************************************************
    */

//define the array of leds
CRGB leds[NUM_LEDS];


//set up button booleans
//set up booleans for buttons to toggle
int buttons[] = {0, 0, 0, 0, 0, 0, 0, 0};
int count = 0;
const int servomagpin = 9;
const int lidservopin = 10;

Adafruit_MotorShield AFMS = Adafruit_MotorShield();
Adafruit_DCMotor *leftMotor = AFMS.getMotor(1);
Adafruit_DCMotor *rightMotor = AFMS.getMotor(2);

Servo servomag;
Servo lidservo;


int mpos = 0;
int lpos = 0;



/*
```

```
    ************************************************************************
    */
/*!
    @brief  Sets up the HW an the BLE module (this function is called
            automatically on startup)
*/
/*
    ************************************************************************
    */
void setup(void)
{
  while (!Serial);  // required for Flora & Micro
  delay(500);

  Serial.begin(115200);
  Serial.println(F("Adafruit␣Bluefruit␣App␣Controller␣Example"));
  Serial.println(F("-----------------------------------------"));

  /* Initialise the module */
  Serial.print(F("Initialising␣the␣Bluefruit␣LE␣module:␣"));

  if ( !ble.begin(VERBOSE_MODE) )
  {
    error(F("Couldn't␣find␣Bluefruit,␣make␣sure␣it's␣in␣CoMmanD␣mode␣&␣
       check␣wiring?"));
  }
  Serial.println( F("OK!") );

  if ( FACTORYRESET_ENABLE )
  {
    /* Perform a factory reset to make sure everything is in a known
       state */
    Serial.println(F("Performing␣a␣factory␣reset:␣"));
    if ( ! ble.factoryReset() ){
      error(F("Couldn't␣factory␣reset"));
    }
  }


  /* Disable command echo from Bluefruit */
  ble.echo(false);

  Serial.println("Requesting␣Bluefruit␣info:");
  /* Print Bluefruit information */
  ble.info();

  Serial.println(F("Please␣use␣Adafruit␣Bluefruit␣LE␣app␣to␣connect␣in␣
     Controller␣mode"));
  Serial.println(F("Then␣activate/use␣the␣sensors,␣color␣picker,␣game␣
     controller,␣etc!"));
  Serial.println();

  ble.verbose(false);  // debug info is a little annoying after this
     point!

  /* Wait for connection */
  while (! ble.isConnected()) {
      delay(500);
  }

  Serial.println(F("******************************"));

  // LED Activity command is only supported from 0.6.6
  if ( ble.isVersionAtLeast(MINIMUM_FIRMWARE_VERSION) )
  {
    // Change Mode LED Activity
    Serial.println(F("Change␣LED␣activity␣to␣" MODE_LED_BEHAVIOUR));
    ble.sendCommandCheckOK("AT+HWModeLED=" MODE_LED_BEHAVIOUR);
  }

  // Set Bluefruit to DATA mode
  Serial.println( F("Switching␣to␣DATA␣mode!") );
  ble.setMode(BLUEFRUIT_MODE_DATA);

  Serial.println(F("******************************"));
/*
    ***************************************************************************
    */
  //LED Strip Setup
  FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
  for(int x = 0; x < NUM_LEDS; x++)
  {
    leds[x] = CRGB(0, 0, 0);
    FastLED.show();
  }

/*
    ****************************************************************************
    */

  //motor setup

  AFMS.begin();  // create with the default frequency 1.6KHz
  //AFMS.begin(1000);  // OR with a different frequency, say 1KHz

  // Set the speed to start, from 0 (off) to 255 (max speed)
  rightMotor->setSpeed(200);
  rightMotor->run(FORWARD);
  // turn on motor
  rightMotor->run(RELEASE);

  leftMotor->setSpeed(200);
  leftMotor->run(FORWARD);
  // turn on motor
  leftMotor->run(RELEASE);

//  servomag.attach(servomagpin);
//  lidservo.attach(lidservopin);
//  servomag.write(90);

}

/*
   *************************************************************************
   */
/*!
    @brief  Constantly poll for new command or response data
*/
/*
   *************************************************************************
   */


void loop(void)
{
  /* Wait for new data to arrive */
  uint8_t len = readPacket(&ble, BLE_READPACKET_TIMEOUT);
//  if (len == 0) return;

  /* Got a packet! */
  // printHex(packetbuffer, len);
  // Buttons
  if (packetbuffer[1] == 'B') {
    uint8_t buttnum = packetbuffer[2] - '0';
    boolean pressed = packetbuffer[3] - '0';
    Serial.print ("Button␣"); Serial.print(buttnum);
    if (pressed)
    {
      if(buttnum == 1)      //pulse blue a few times
      {
        for(int z = 0; z < 4; z++)
        {
          for(int x = 0; x < 100; x++)
          {
             for(int y = 0; y < NUM_LEDS; y++)
             {
              leds[y] = CRGB(0, 0, x);
             }
            FastLED.show();
            delay(4);
          }

          for(int x = 100; x >= 0; x--)
          {
             for(int y = 0; y < NUM_LEDS; y++)
             {
              leds[y] = CRGB(0, 0, x);
             }
            FastLED.show();
            delay(4);
          }
          delay(300);
        }
      }
      else if(buttnum == 2)          //pulse yellow
      {
        for(int z = 0; z < 4; z++)
        {
          for(int x = 0; x < 100; x++)
          {
             for(int y = 0; y < NUM_LEDS; y++)
             {
              leds[y] = CRGB(x, x, 0);
             }
            FastLED.show();
            delay(4);
          }

          for(int x = 100; x >= 0; x--)
          {
           for(int y = 0; y < NUM_LEDS; y++)
             {
              leds[y] = CRGB(x, x, 0);
             }
            FastLED.show();
            delay(4);
          }
          delay(300);
        }
      }
      else if(buttnum == 3)      //plant happy pulse green
      {
```

```
      for(int z = 0; z < 4; z++)
      {
        for(int x = 30; x < 150; x++)
        {
           for(int y = 0; y < NUM_LEDS; y++)
           {
            leds[y] = CRGB(0, x, 0);
           }
          FastLED.show();
          delay(10);
        }

        for(int x = 150; x >= 30; x--)
        {
         for(int y = 0; y < NUM_LEDS; y++)
           {
            leds[y] = CRGB(0, x, 0);
           }
          FastLED.show();
          delay(10);
        }
      }

      for(int x = 30; x >= 0; x--)
      {
       for(int y = 0; y < NUM_LEDS; y++)
         {
          leds[y] = CRGB(0, x, 0);
         }
        FastLED.show();
        delay(10);
      }

      delay(30);
    }
    else if(buttnum == 4)       //detach from plant
    {
      //rightMotor->run(BACKWARD);
      //leftMotor->run(BACKWARD);
      //delay(500);
      rightMotor->run(RELEASE);
      leftMotor->run(RELEASE);
    }

    else if(buttnum == 5 || buttnum == 6 || buttnum ==7 || buttnum ==
        8)
    {
      if (buttnum == 5)        //this is button 5, forward
```

```
        {
            rightMotor->run(BACKWARD);
            leftMotor->run(FORWARD);
        }
        if (buttnum == 6)         //button 6, backwards
        {
            rightMotor->run(FORWARD);
            leftMotor->run(BACKWARD);
        }
        if (buttnum == 7)          //button 7 turn left
        {
            rightMotor->run(BACKWARD);
            leftMotor->run(BACKWARD);
        }
        if (buttnum == 8)           //button 8 turn right
        {
            rightMotor->run(FORWARD);
            leftMotor->run(FORWARD);
        }
      }
      else
      {
        rightMotor->run(RELEASE);
        leftMotor->run(RELEASE);
      }



    }
    else
    {
      Serial.println("␣released");
    }
  }

}
```