Hiking Buddy

Group 42:
Joshua Ax, Jonathan Doyle, Ryan Harty, Nathan Nash

MECH307: Mechatronics and Measurement Systems

Colorado State University

December 9th, 2016

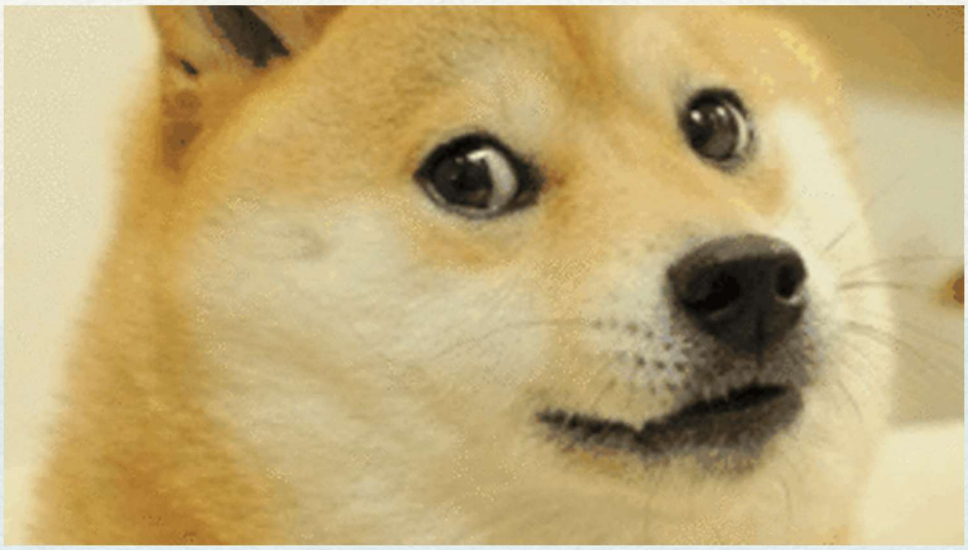
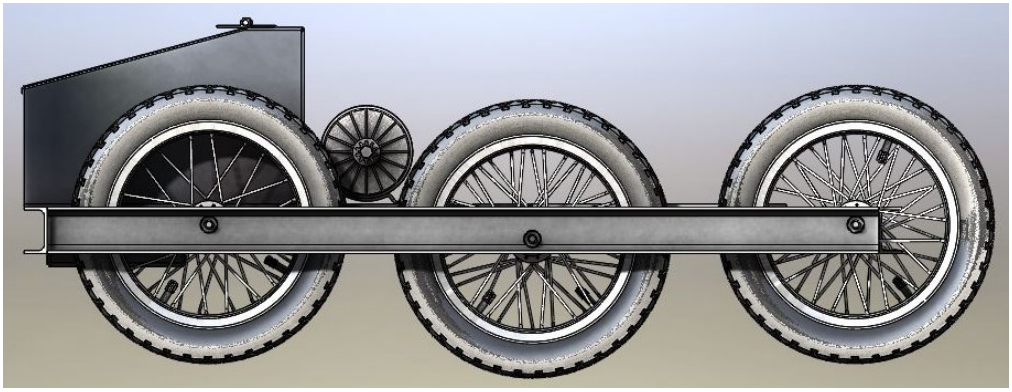


Table of Contents

Design Summary

The Hiking Buddy is a companion robot that accompanies the user while they hike a light to medium difficulty trail. The Hiking Buddy autonomously follows the user while providing useful geolocation information. The user is able to enjoy a reduced backpack load due to the Hiking Buddy's large carrying capacity. A wireless key chain remote enables control of the Hiking Buddy's follow states including a run state, a hold state, and the ability to increase and decrease the following distance.



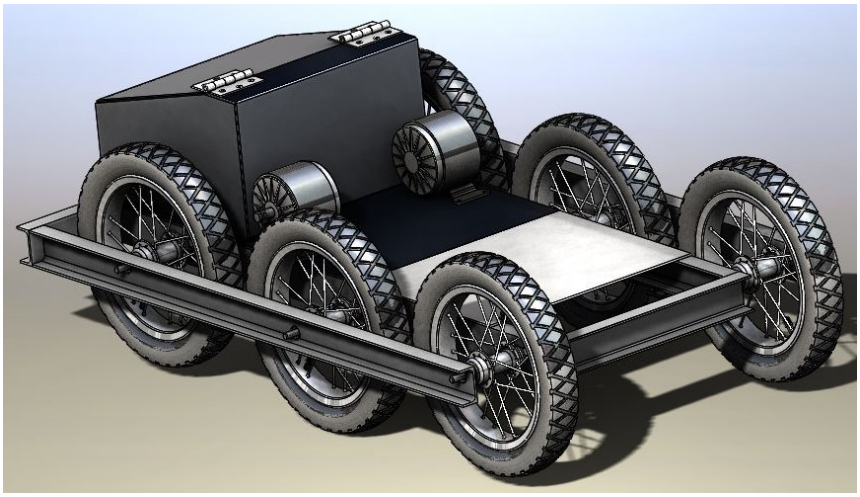


Figure 1 & 2: Early 3D models of the Hiking Buddy.

The Hiking Buddy's ability to follow a hiker autonomously is provided by an object tracking camera called Pixy. The Pixy is programmed to identify and track the hiker's vibrant monotone jacket. The solid vibrant color of the hiker's jacket easily allows the Pixy to distinguish the hiker from other hikers and objects. Hiking Buddy is able to estimate how far away and where the hiker is based on the number and location of pixels that the hiker's jacket occupies in the camera's view. This information is used to calculate the angular velocity at which the motors should run to keep Hiking Buddy close to the hiker. When the hiker goes off center of the camera frame, the Pixy's servos will realign the camera with the hiker. The change in yaw is used to calculate how much the left or right drive train should change speed to reposition the Hiking Buddy to the hiker.

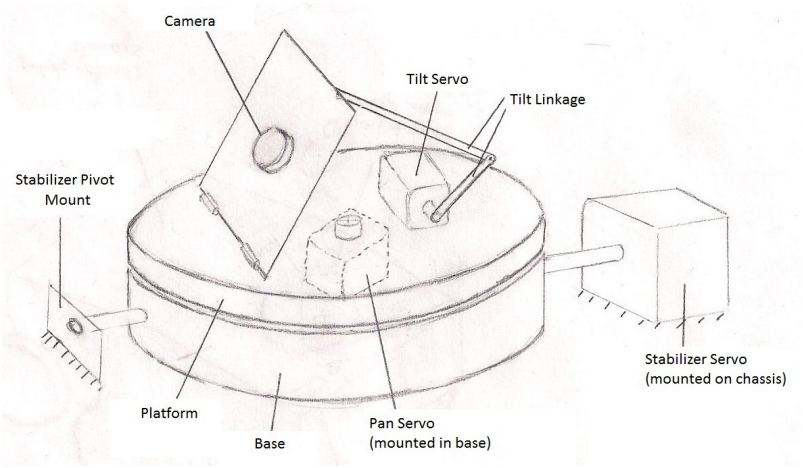


Figure 3: Pan tilt stabilizer for the Pixy camera.

The key fob allows the Hiking Buddy's motor systems to be controlled remotely. The key fob is able to instantly freeze the motors. This is useful in the case that the camera begins tracking an object other than the hiker. To help prevent tight corners on the trail from becoming problematic, the follow distance can be can be changed via the key fob. The user can decrease the follow distance when coming to a tight corner to keep the Hiking Buddy from going off trail. Greater following distances allow the hiker to enjoy nature with the Hiking Buddy following far off in the distance. This remote system provides control from a distance once the Hiking Buddy has been turned on and adjusted for the lighting conditions.

System Details

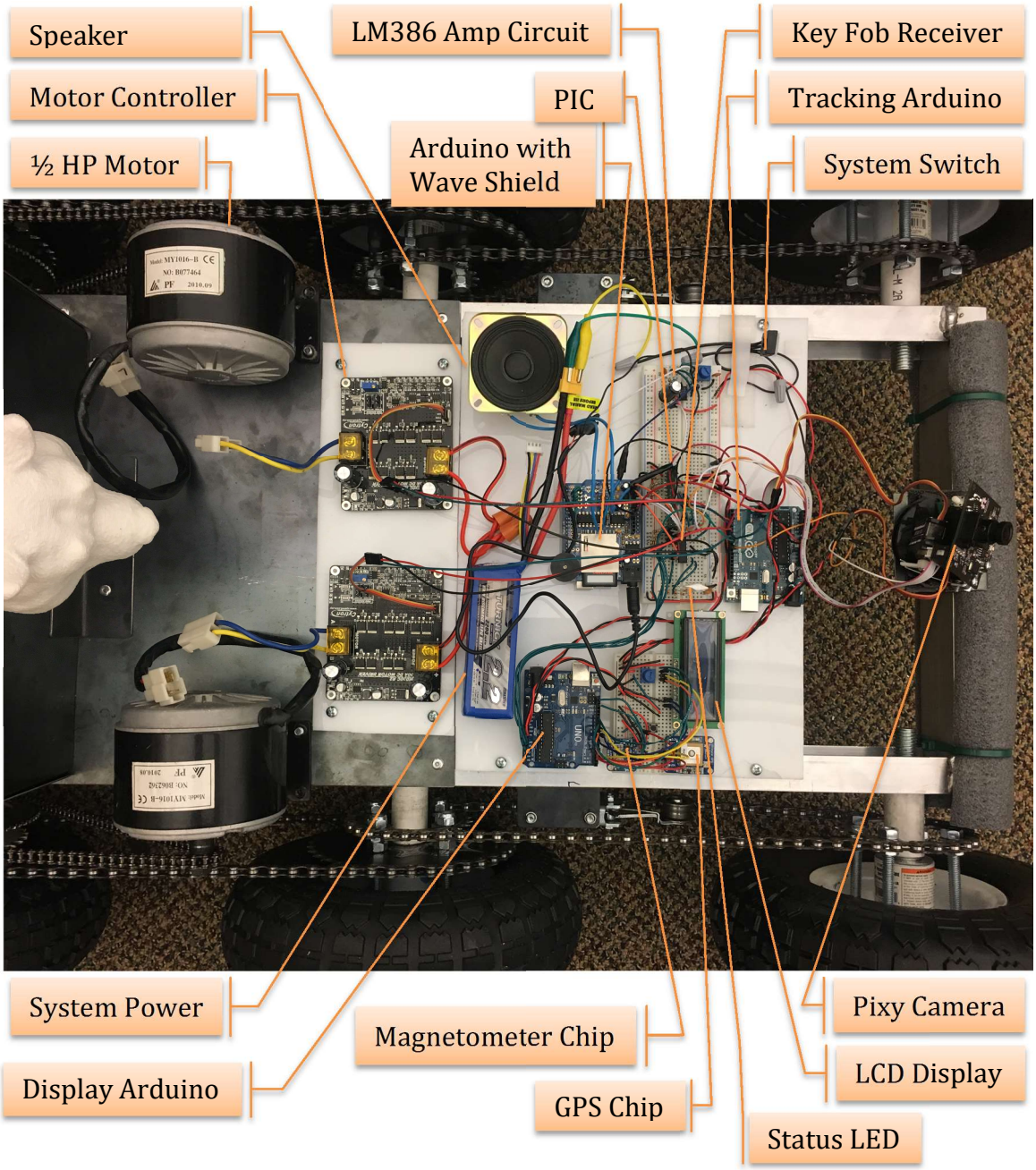


Figure 4: Hiking Buddy's electrical system configuration.

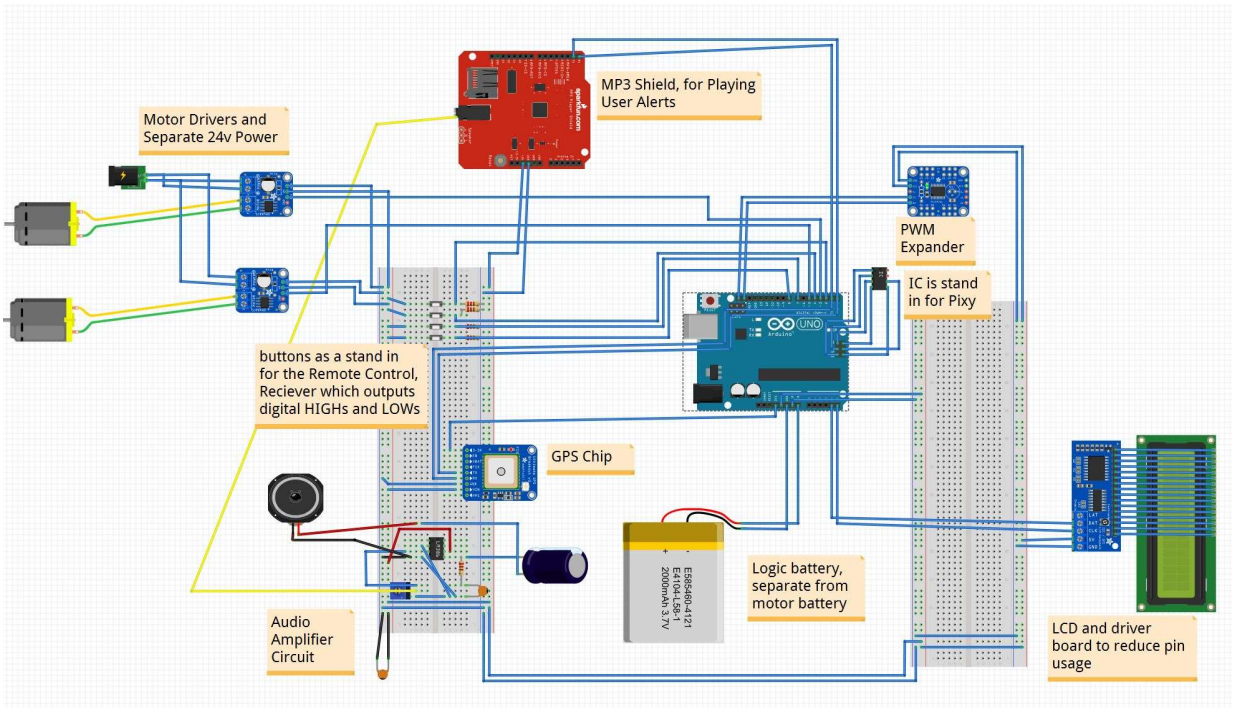


Figure 5: Hiking Buddy's wiring diagram.

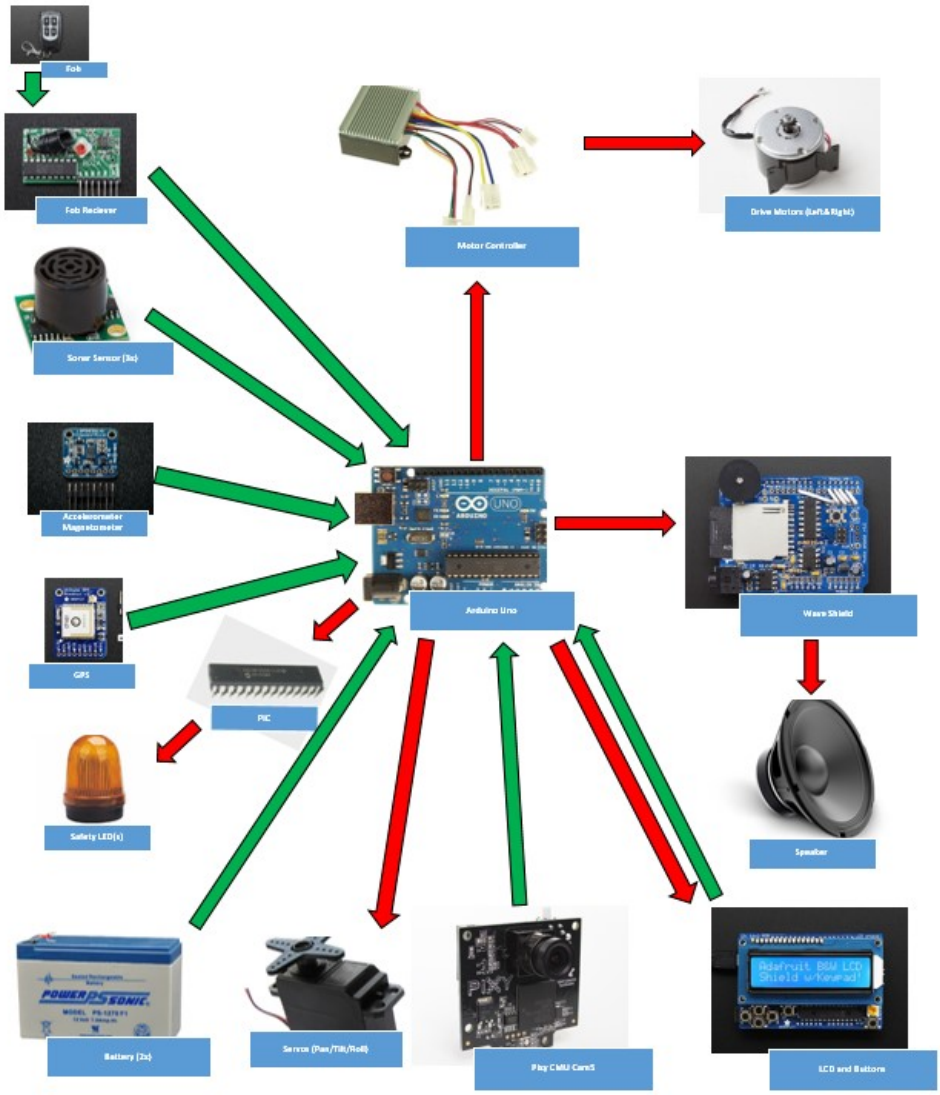


Figure 6: Hiking Buddy's functional diagram.

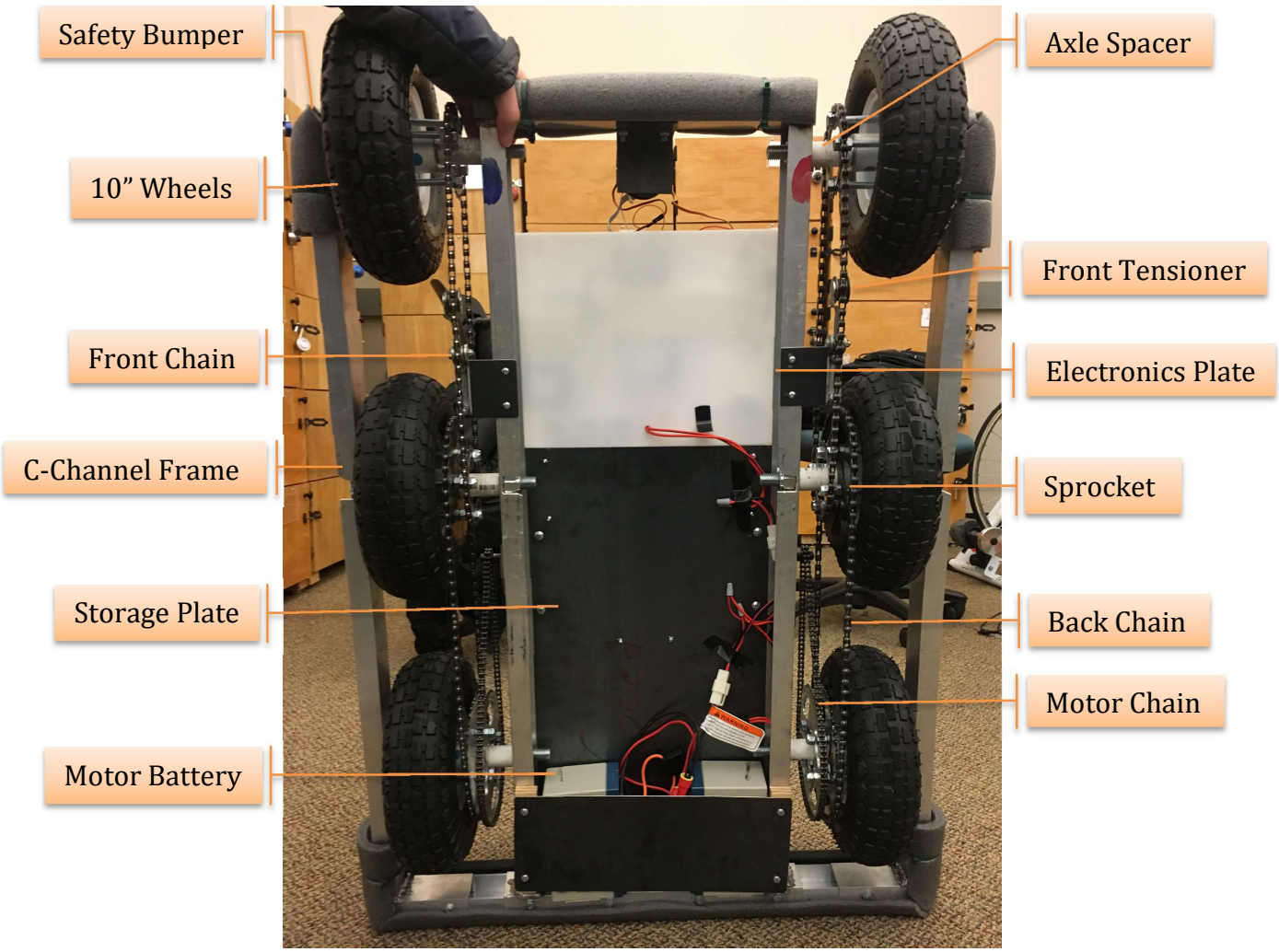


Figure 7: Hiking Buddy’s chassis and drive train system configuration.

Chassis

The main structural portion of the chassis is constructed from welded 2”x1” aluminum c-channel. The wheel axle nuts are tightened onto the axle bolts which gives the chassis torsional stability. This provides enough rigidity so that no brackets are needed between the wheels. The middle axle is dropped .65” lower than the front and rear wheels. This means that only 4 out of the 6 wheels are simultaneously in contact with the ground, decreasing the power required to turn the Hiking buddy. The storage compartment is made from 16-gauge steel sheet metal and is mounted to the storage plate that spans the central section on the chassis. The batteries to power the motor are mounted beneath the storage plate for better aesthetics. All the electronics are mounted on high-density polyethylene board, which isolates the electronics and prevents them from accidently bridging across the sheet metal. Foam padding was added to the front, back, and corners of the frame for safety purposes.

Drive Train

Each bank of wheels is driven by a ½ horsepower electric motor. The motors are chained to the back wheels through a 1:7 gear reduction. This gear reduction allows for the Hiking Buddy to have a top speed of just over 12 mi/hr. The chain system from the back to middle and middle to front are spare parts from bikes and have a gear ratio of 1:1. All the sprockets are aligned and mounted using the same bolts that hold the wheel hubs together. The motor sprockets, motors, motor chains, motor batteries, and front tensioner are from two used kids' mopeds.

Output Display

The Hiking Buddy has an LCD output display which is utilized to display the robot's GPS coordinates, elevation, and magnetic heading. The Hiking Buddy uses a GPS chip and a magnetometer chip to generate these display inputs.

Audio Output

The Hiking Buddy has an audio amplifier circuit that allows it to play any number of audio files. The circuit uses an LM386 audio amplifier integrated circuit with a user adjustable potentiometer allowing the user to adjust the volume to their needs. The amplifier circuit receives a signal input from an MP3 wave shield which stores preset audio files. The wave shield plays a different message through the amplifier circuit based off of an input from the Pic. Whenever the following distance is increased or decreased a corresponding audio message is played through the speaker to indicate to the user the following state has been changed. Additionally, when the run state is changed to either 'run' or 'hold' an audio message is played informing the user. Finally, if the robot locks on to a target or loses its target it will inform the user by playing a corresponding audio message.

Sensors

The Hiking Buddy's main sensor is the Pixy camera. The Pixy camera tracks the user via color recognition. In addition, the Hiking Buddy has sensors for acceleration and magnetometer head, as well as GPS coordinates.

Design Evaluation

The Hiking meets or exceeds the requirements set forth for each functional category, as outlined below.

Output Display

I. LCD Display
   a. Displays a welcome screen on startup and GPS coordinates, altitude, compass heading, following distance, and running mode in five user-selectable menus.

II. Status Light RGB LED
   a. Indicates whether the robot is in run mode, wait mode, looking for the hiker, or starting up. The LED status light is very useful when you want to know what the robot is trying to do.

The LCD displays "No GPS fix." when the GPS chip cannot get a signal from a satellite. This problem is likely to occur when indoors, but satellites are usually available when outdoors. All the output displays work properly in hiking conditions and provide relevant and useful information.

Audio Output Device

I. State Driven Messages
   a. A pre-recoded message plays though the speaker whenever the robot's status changes.

II. Higher volume via an LM386 audio amplifier circuit.

The messages are stored on an SD card and played through the LM386 amplifier circuit and a 2.5" speaker using the audio Arduino with the Wave Shield. More details are below in the Logic, Processing, and Control section.

Manual User Input

I. Radio Frequency Key Fob
   a. Allows the user to pause/run the robot or increase/decrease the following distance remotely.

II. Display Change Button
   a. A push button allows the user to switch between five menus on the LCD

III. Output Adjustment Potentiometers
   a. Display contrast adjustment
   b. Volume knob on Wave Shield
   c. Audio amplifier gain adjustment

Sensors

I. Pixy Camera
   a. Tracks the hikers jacket through color image recognition.
II. GPS Chip
   a. Collects geolocation information.
III. Magnetometer
   a. Collects magnetic field information for determining compass heading.

Actuators, Mechanisms, and Hardware

I. Electric Motors
   a. Two 1/2hp motors give the robot enough power to handle rough terrain.
II. Chain Tensioners
   a. Prevents the front two chains from coming off the sprockets.
III. Storage Box
   a. Large storage box made out of 16-gauge steel stores the tracking jacket and whatever else the hiker desires.
IV. Aluminum Chassis
   a. Welded out of 1”x2’ aluminum c-channel that doesn’t flex on rough terrain.

Logic, Processing, and Control; AND Miscellaneous

I. Pixy Camera
   a. Takes video from an adjustable NTSF camera and processes it looking for pixels within a tolerance of a preset color
   b. If a block larger than 5x5 pixels is seen the x and y coordinates of the center point as well as the width and height are outputted through the ICSP pins
   c. New coordinates and height and width are sent to the Arduino are sent at a rate of 50Hz
II. Main Drive Arduino
   a. Takes in x and y coordinates of the object and uses their distance from the center of frame to drive the servos on the pan tilt mount to keep the object in center frame
   b. The outputted height of the object is then compared against the desired height of the object, adjusted by the distance adjustment factor sent from the PIC, and used to set the forward speed of the drive motors.
   c. The position of the pan servo is used to derive a turn factor between negative 150 and 150 which is subtracted from the previously derived speed of the right motor and added to the speed right motor.
   d. The state of the running mode pin attached to the PIC is checked, if the pin is high the motors are driven, if not the motors are held in brake mode
III. PIC 16F88

a. Tracks the state of the RF receiver pins and "seeing" state from Main Drive Arduino.
    i. Toggles running state between "run" and "hold" based on key fob input and whether the camera is seeing the object.
    ii. Increments and decrements following distance levels between 0 and 7.
b. Outputs running state.
c. Outputs following distance level in binary using three pins.
d. Communicates initialization state, "running" state, "seeing" state, and temporary object search to the user by driving the status LED with solid colors and flash patterns.

IV. LCD Arduino
a. Receives GPS and magnetometer information using serial communication.
    i. Parses GPS sentences to extract relevant data.
    ii. Converts magnetometer data into compass heading in degrees and as a character string.
b. Tracks "run" state and follow distance values.
c. Displays a welcome message, GPS coordinates, altitude, compass heading, running state, and following distance in five user-selectable menus.

V. Audio Arduino with Wave Shield
a. Tracks "run" state, "seeing" state, and follow distance value.
b. Plays pre-recorded messages to inform the hiker when the Hiking Buddy has changed between "run" and "hold" mode, when the object is lost or found by the Pixy camera, and when the following distance is increased or decreased.

Partial Parts List

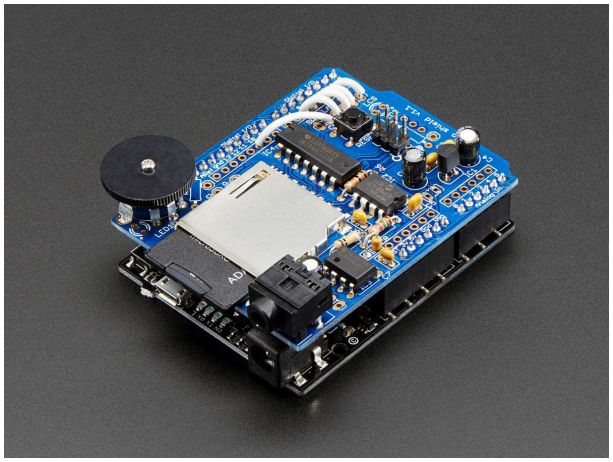
½ HP 30 Amp Motors

Scavenged from used Razor Pocket Mod scooters, Model Number MY1016, $36.99 retail

Pixy CMUCam5

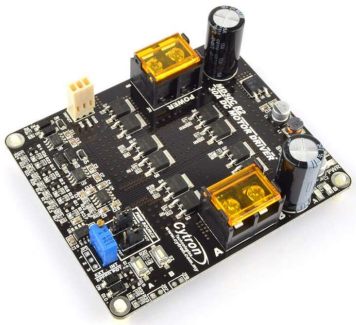
From adafruit.com, Product ID 1906, $74.95

Adafruit Wave Shield



From adafruit.com, Product ID 94, $22.00

Cytron 30A 5-30V Single Brushed DC Motor Driver

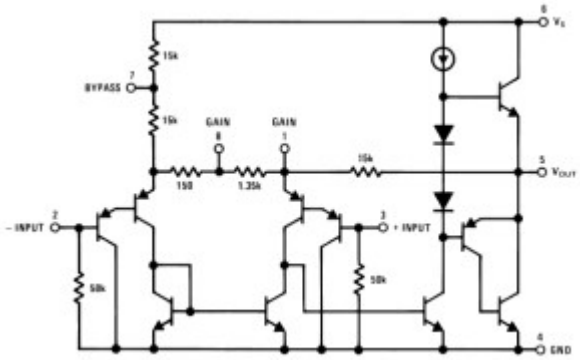


From robotshop.com, Product Code RB-Cyt-133, $33.13

Simple RF M4 Receiver

Momentary key fob receiver, pins are pulled high when key fob buttons are pushed. From adafruit.com, $4.95

LM386 Audio Amplifier



From Mountain State Electronics, $1.99

Lessons Learned

Every project is full of lessons learned, the lessons our group learned are detailed below.

1. Start early to avoid cramming to complete your project at the last minute.
    a. For the most part our group started building the Hiking Buddy over fall break. We had to complete a lot of manufacturing work including metal working, welding, and machining which always takes more time than you realize. Additionally, many times things don't work the first time you try them out and could be from a number of reasons. Our first motor controllers from the kids' moped didn't work. We had to wait a couple days while new ones were shipped to us wasting valuable time.

2. Plan ahead for parts that will need time to ship.
    A week before the project was due we went to order sprockets with 2-day shipping and found out the you not arrive till the day the project was due. We ended up going to multiple used bike stores scavenging for matching sprockets. Even if you can order an item with 2-day shipping, plan on it taking at least a week.

3. Things will break, so improvise and bring tools/extras.
    24 hours before the lab presentation we broke a servo. We had a replacement that we had to attach with zip ties. An hour before the class presentation we broke a servo mount. Thankfully we had super glue and zip ties on us and were able to fix it. During the presentation we blew a fuse. This happened before so we brought some with us and were able to fix it in minutes. Keep common tools, duct tape, glue, etc. on hand. If it has broken once before, it most likely will break again so have extras.

4. Test the device in the place you will be presenting it.
    We ended up blowing a fuse because the motors pulled too much current. We had never tested the robot on carpet, which was the flooring for the class presentation. We replaced the fuse and demonstrated it in the hall.

5. Give yourself extra room for play when designing a part.
    We designed and welded a frame and later found out the new wheels we chose wouldn't fit in the wheel wells. We cut the frame apart and mounted it back together with extra material we had. Lesson learned, you can always take away material and it's always harder to add material.

Appendix

Calculations

Top speed:

$$v = \frac{\pi(\omega_{\max})(D_w)(\mathrm{D}_m)}{D_s}$$

$$v = \frac{3.14(2850rpm)(10\,\mathrm{in})(0.9\,\mathrm{in})}{6.32in}$$

$$v = 12743.83in/\min = 12.068mph$$

Code

Audio:

```
/*
   Audio code
   Gathers distance, running, and seeing status and
   outputs audio messages when statuses change
*/

#
include < FatReader.h > #include < SdReader.h > #include < avr /
pgmspace.h > #include "WaveUtil.h"#
include "WaveHC.h"

SdReader card; // This object holds the information for the card
FatVolume vol; // This holds the information for the partition on the
card
FatReader root; // This holds the information for the filesystem on the
card
FatReader f; // This holds the information for the file we're play

WaveHC wave;

boolean running = 0;
boolean seeing = 0;
boolean obstacle = 0;
byte distance = 4;
boolean running_old = 0;
boolean seeing_old = 0;
boolean obstacle_old = 0;
byte distance_old = 4;

byte dist_pin_0 = A0;
byte dist_pin_1 = A1;
byte dist_pin_2 = A2;
byte running_pin = A3;
```

```
byte seeing_pin = A4;
byte obstacle_pin = A5;

//returns the number of bytes currently free in RAM for debugging
int freeRam(void) {
  extern int __bss_end;
  extern int * __brkval;
  int free_memory;
  if ((int) __brkval == 0) {
    free_memory = ((int) & free_memory) - ((int) & __bss_end);
  } else {
    free_memory = ((int) & free_memory) - ((int) __brkval);
  }
  return free_memory;
}

void sdErrorCheck(void) {
  if (!card.errorCode()) return;
  putstring("\n\rSD I/O error: ");
  Serial.print(card.errorCode(), HEX);
  putstring(", ");
  Serial.println(card.errorData(), HEX);
  while (1);
}

void setup() {
  pinMode(dist_pin_0, INPUT);
  pinMode(dist_pin_1, INPUT);
  pinMode(dist_pin_2, INPUT);
  pinMode(running_pin, INPUT);
  pinMode(seeing_pin, INPUT);
  //pinMode(obstacle_pin, INPUT);

  byte i;

  // set up serial port
  Serial.begin(9600);
  putstring_nl("WaveHC");

  putstring("Free RAM: "); //assists with debugging
  Serial.println(freeRam()); // keep over 150 bytes

  // Set the output pins for the DAC control. This pins are defined in
the library
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);

  if (!card.init()) {
    putstring_nl("Card init. failed!"); // Something went wrong, print
out why
    sdErrorCheck();
```

```
    while (1); // wait indefinitely
  }

  // look for a FAT partition
  uint8_t part;
  for (part = 0; part < 5; part++) { // we have up to 5 slots to look
in
    if (vol.init(card, part))
      break; // found one
  }
  if (part == 5) { // if we ended up not finding one
    putstring_nl("No valid FAT partition!");
    sdErrorCheck(); // Something went wrong, print out why
    while (1); // wait indefinitely
  }

  // Lets tell the user about what we found
  //putstring("Using partition ");
  //Serial.print(part, DEC);
  //putstring(", type is FAT");
  //Serial.println(vol.fatType(),DEC);     // FAT16 or FAT32?

  // Try to open the root directory
  if (!root.openRoot(vol)) {
    putstring_nl("Can't open root dir!"); // Something went wrong,
    while (1); // wait indefinitely
  }

  // Whew! We got past the tough parts.
  putstring_nl("Ready!");

  delay(2000); //allow other controllers to boot up before checking for
info
}

SIGNAL(TIMER2_OVF_vect) {}

void loop() {
  byte i;
  static byte playing = -1;

  //assign value to distance using pins as 3 least significant bits
  distance = digitalRead(dist_pin_0) | digitalRead(dist_pin_1) << 1 |
digitalRead(dist_pin_2) << 2;

  running = digitalRead(running_pin);
  seeing = digitalRead(seeing_pin);
  //obstacle = digitalRead(obstacle_pin);

  if (running > running_old) {
    if (playing != 0) {
      playing = 0;
      playfile("RUN.WAV");
```

```
      Serial.println("run");
    }
  } else if (running < running_old) {
    if (playing != 1) {
      playing = 1;
      playfile("STAY.WAV");
      Serial.println("stay");
    }
  } else if (distance > distance_old) {
    if (playing != 2) {
      playing = 2;
      playfile("FURTHER.WAV");
      Serial.println("further");
    }
  } else if (distance < distance_old) {
    if (playing != 3) {
      playing = 3;
      playfile("NEARER.WAV");
      Serial.println("nearer");
    }
  } else if (seeing > seeing_old) {
    if (playing != 4) {
      playing = 4;
      playfile("FOUND.WAV");

      Serial.println("found");
    }
  } else if (seeing < seeing_old) {
    if (playing != 5) {
      playing = 5;
      playfile("LOST.WAV");
      Serial.println("lost");
    }
  } else if (obstacle == 1 && obstacle_old == 0) {
    if (playing != 5) {
      playing = 5;
      playfile("OBSTACLE.WAV");
      Serial.println("obstacle");
    }
  }

  distance_old = distance;
  seeing_old = seeing;
  running_old = running;
  //obstacle_old = obstacle;

  if (!wave.isplaying) {
    playing = -1;
  }
}

void playfile(char * name) {
  // see if the wave object is currently doing something
```

```
  if (wave.isplaying) { // already playing something, so stop it!
    wave.stop(); // stop it
  }
  // look in the root directory and open the file
  if (!f.open(root, name)) {
    putstring("Couldn't open file ");
    Serial.print(name);
    return;
  }
  // read the file and turn it into a wave object
  if (!wave.create(f)) {
    putstring_nl("Not a valid WAV");
    return;
  }
  Serial.print("playing");
  Serial.println(name);
  // start playback
  wave.play();
}
```

LCD:

```
/*
    LCD code
    Gathers heading, lat/long, and altitude data
    from accelerometer/magnetometer and gps module
    and displays on LCD with button menu cycling
*/

//libraries for accelerometer/magnetometer
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>

//libraries for gps
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>

//library for lcd
#include <LiquidCrystal.h>


//initialize button variables
boolean buttonState = 0;
boolean last_buttonState = 0;
boolean buttonPressed = 0;
unsigned long changeTime = 0;
byte buttonPin = 4;
byte debounceDelay = 10;

byte dist_pin_0 = A0;
byte dist_pin_1 = A1;
byte dist_pin_2 = A2;
byte running_pin = A3;

boolean running = 0;
byte distance = 0;
boolean running_old = 0;
byte distance_old = 5;

byte menu = 1;

uint32_t timer = millis();

//initialize lcd with numbers of interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

//software serial communication for gps
SoftwareSerial mySerial(3, 2);

Adafruit_GPS GPS(&mySerial);
```

```
// Assign a unique ID to accelerometer/magnetometer
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);


void setup(void)
{

  pinMode(dist_pin_0, INPUT);
  pinMode(dist_pin_1, INPUT);
  pinMode(dist_pin_2, INPUT);
  pinMode(running_pin, INPUT);
  pinMode(buttonPin,INPUT);

  //initialize lcd with number of columns, rows
  lcd.begin(16, 2);

  //display welcome screen on startup
  lcd.setCursor(0, 0);        //column, row
  lcd.print(" *RoboDoge 1.0* ");
  lcd.setCursor(0, 1);        //column, row
  lcd.print("Hello Human!    ");

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some
use 4800
  GPS.begin(9600);

  //turn on RMC (recommended minimum) and GGA (fix data) including
altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the
data, and
  // print it out we don't suggest using anything higher than 1 Hz

  // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

  // GPS data is read on a 1ms interrupt
  OCR0A = 0xAF;
  TIMSK0 |= _BV(OCIE0A);

  // Initialise the accelerometer/magnetometer
  mag.begin();

  //parse GPS sentences into variables
  GPS.parse(GPS.lastNMEA());

  //wait for button to be pressed
  while(!buttonPressed){
    check_button();
  }
```

```
}


// Interrupt is called once a millisecond, looks for any new GPS data,
and stores it
SIGNAL(TIMER0_COMPA_vect) {
  char c = GPS.read();
}

void refresh_display(){
  if(menu == 1) disp_heading();
  if(menu == 2) disp_lat_long();
  if(menu == 3) disp_altitude();
  if(menu == 4) disp_distance();
  if(menu == 5) disp_mode();
}

void disp_heading(){
  //Get a new sensor event
  sensors_event_t event;
  mag.getEvent(&event);

  //float Pi = 3.14159;

  // Calculate the angle of the vector y,x
  // Declination at zip code 80521 is approx 8.5 degrees
  float deg = ((atan2(event.magnetic.y,event.magnetic.x) * 180) /
3.14159) + 8.5;

  // Normalize to 0-360
  if (deg < 0)
  {
    deg = 360 + deg;
  }

  //convert degrees heading to string of two characters
  char heading[1];
  if(deg >= 247.5 && deg < 292.5)
  {
    heading[0] = 'W';
    heading[1] = ' ';
  }
  else if(deg >= 67.5 && deg < 112.5)
  {
    heading[0] = 'E';
    heading[1] = ' ';
  }
  else
  {
    if(deg >= 292.5 || deg < 67.5)
    {
      heading[0] = 'N';
    }
```

```
    else if(deg >= 112.5 && deg < 247.5)
    {
      heading[0] = 'S';
    }
    if(deg >= 202.5 && deg < 337.5)
    {
      heading[1] = 'W';
    }
    else if(deg >= 22.5 && deg < 157.5)
    {
      heading[1] = 'E';
    }
    else
    {
      heading[1] = ' ';
    }
  }


  lcd.clear();
  lcd.setCursor(0, 0);        //column, row
  lcd.print("Heading: ");
  lcd.print(heading[0]);      //first heading character
  lcd.print(heading[1]);      //second heading character
  lcd.setCursor(0, 1);        //column, row
  lcd.print(deg,0);           //prints heading in degrees
  lcd.print((char)223);       //degree symbol
  lcd.print(" CW from N");
}

void disp_lat_long(){
  if(GPS.fix){
    lcd.clear();
    lcd.setCursor(0, 0);                    //column, row
    lcd.print("Lat    ");
    lcd.print(GPS.latitudeDegrees, 5);
    lcd.print((char)223);                   //degree symbol
    lcd.setCursor(0, 1);                    //column, row
    lcd.print("Long ");
    lcd.print(GPS.longitudeDegrees, 5);
    lcd.print((char)223);                   //degree symbol
  }
  else
  {
    lcd.clear();
    lcd.setCursor(0, 0);                    //column, row
    lcd.print("No GPS fix.");
  }
}

void disp_altitude(){
  if(GPS.altitude < 1){
    lcd.clear();
```

```
    lcd.setCursor(0, 0);                    //column, row
    lcd.print("No altitude fix.");
    lcd.setCursor(0, 1);                    //column, row
  }
  else
  {
    lcd.clear();
    lcd.setCursor(0, 0);                     //column, row
    lcd.print("Altitude: ");
    lcd.print(GPS.altitude*3.28084, 0);      //converts meters to feet
    lcd.print("'");
    lcd.setCursor(0, 1);                     //column, row
  }
}

void disp_distance(){

  //assign value to distance using pins as 3 least significant bits
  distance = digitalRead(dist_pin_0) | digitalRead(dist_pin_1) << 1 |
digitalRead(dist_pin_2) << 2;
  distance ++;

  lcd.clear();
  lcd.setCursor(0, 0);                    //column, row
  lcd.print("Follow Distance:");
  lcd.setCursor(0, 1);                    //column, row
  lcd.print(distance);
  if(distance == 1){
    lcd.print(" - Closest");
  }else if(distance == 8){
    lcd.print(" - Farthest");
  }
}

void disp_mode(){

  running = digitalRead(running_pin);

  if(running == 1){
    lcd.clear();
    lcd.setCursor(0, 0);                    //column, row
    lcd.print("Running mode!");
    lcd.setCursor(0, 1);                    //column, row
    lcd.print("Press A to hold");
  }
  else
  {
    lcd.clear();
    lcd.setCursor(0, 0);                    //column, row
    lcd.print("Holding mode");
    lcd.setCursor(0, 1);                    //column, row
    lcd.print("Press B to run");
  }
```

```
}

void check_button(){
  boolean pin_read = digitalRead(buttonPin);

  // if millis() or changeTime wraps around, reset it
  if (changeTime > millis())  changeTime = millis();

  if(pin_read != last_buttonState)                  //check if button
state has changed
  {
    changeTime = millis();                         //log time of change
  }
  else if(millis() - changeTime > debounceDelay)   //if button has been
at same state
  {                                                //for longer than
debounce delay
    if(pin_read != buttonState)             //if it has changed state
    {
      buttonState = pin_read;
      if(buttonState){
        buttonPressed = buttonState;     //buttonPressed is true if
buttonState has changed from low to high
      }
    }
  }
  last_buttonState = pin_read;
}

void loop()
{
  //parse GPS sentences into variables
  GPS.parse(GPS.lastNMEA());

  // if millis() or timer wraps around, reset it
  if (timer > millis())  timer = millis();

  // refresh the display every 2 seconds
  if (millis() - timer > 2000) {
    timer = millis(); // reset the timer
    refresh_display();
  }

  //check button state continuously
  check_button();

  //check for running and distance state changes
  if(distance != distance_old){
    menu = 4;
    distance_old = distance;
    refresh_display();
  }
  else if(running != running_old){
```

```
    menu = 5;
    running_old = running;
    refresh_display();
  }


  //cycle through menu if button is pressed
  if(buttonPressed){
    if(menu < 5) menu++;    //cycle menu 1-2-3-1-2-...
    else menu = 1;
    refresh_display();      //refresh immediately upon menu change
    buttonPressed = 0;
  }
}
```

Main:

```
//Include libraries
#include <Adafruit_PWMServoDriver.h>    //library for the pwm expander
#include <Wire.h>                       //library for serial
communication
#include <Adafruit_Sensor.h>            //overarching adafruit sensor
libary
#include <SPI.h>                        //serial communication libary
for the Pixy
#include <Pixy.h>                       //specific pixy library
#include <Servo.h>                      //servo library


//Set control pins for motor controllers
int powerL = 3;
int dirL   = 4;
int powerR = 5;
int dirR   = 7;

//declare variables for motor speed to be modified during tracking
int rSpeed = 0;
int lSpeed = 0;
//declare variable that detirmines by what factor the robot turns
int turn = 0;
//variable that detirmines by what distance the robot follows
int dist = 80;
//variable that detirmines how much closer or further the robot follows
int distAdj = 0;
int distance = 0;

//create pixy object
Pixy pixy;
//variables for values coming out of pixy
int x = 160;
int y = 0;
int width = 0;
int height = 0;
//variable for tracking time since object spotted
uint32_t lastBlockTime = 0;

//create servo objects for pan and tilt servos on the pixy
int panPin = 9;
int tiltPin = 6;

Servo pan;
Servo tilt;

//variables for how much to increment pan/tilt servos
int panSpeed = 0;
int tiltSpeed = 0;
//variables for positions of pan/tilt servos
int panPos = 106;
```

```
int tiltPos = 120;

int panPulse = 400;
int tiltPulse = 400;

//input pins
int rMode = 2;
int distIn0 = A0;
int distIn1 = A1;
int distIn2 = A2;

//data output pins
int seeing = 8;

// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
// you can also call it with a different address you want
//Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x41);

// these are experimentally derived for the pan tilt servos on the
mount
#define PANMIN  150 // this is the 'minimum' pulse length count (out of
4096)
#define PANMAX  550 // this is the 'maximum' pulse length count (out of
4096)

#define TILTMIN  225 // this is the 'minimum' pulse length count (out
of 4096)
#define TILTMAX  550 // this is the 'maximum' pulse length count (out
of 4096)

void setup()
{
  //initialize input pins
  pinMode(rMode, INPUT);
  pinMode(distIn0, INPUT);
  pinMode(distIn1, INPUT);
  pinMode(distIn2, INPUT);

  //initialize data output pins
  pinMode(seeing, OUTPUT);

  //start serial communication for debugging
  Serial.begin(9600);
  //start pixy
  pixy.init();
  //start PWM expander chip
  pwm.begin();

  pwm.setPWMFreq(60);  // Analog servos run at ~60 Hz updates

  pan.attach(panPin);
  tilt.attach(tiltPin);
```

```
  yield();

  pwm.setPWM(panPin, 0, PANMAX / 2);
  pwm.setPWM(tiltPin, 0, TILTMAX / 2);

  //set all motor pins as outputs
  pinMode(dirL, OUTPUT);
  pinMode(powerL, OUTPUT);
  pinMode(dirR, OUTPUT);
  pinMode(powerR, OUTPUT);
  //Set all motors to have no speed
  analogWrite(powerL, 0);
  digitalWrite(dirL, LOW);
  analogWrite(powerR, 0);
  digitalWrite(dirR, LOW);
  delay(1000);
}

////////////START OF PRIMARY DRIVE LOOP////////////
////////////START OF PRIMARY DRIVE LOOP////////////
////////////START OF PRIMARY DRIVE LOOP////////////
////////////START OF PRIMARY DRIVE LOOP////////////

void loop()
{
  //variables for the pixy
  static int i = 0;
  int j;
  uint16_t blocks;
  char buf[32];

  //ask the pixy to tell us what it sees for signature 1
  blocks = pixy.getBlocks();

  //store what the pixy sees in respective variables
  x = pixy.blocks[0].x;
  y = pixy.blocks[0].y;
  width = pixy.blocks[0].width;
  height = pixy.blocks[0].height;

  //if the pixy sees an object
  if (blocks)
  {
    //run pan and tilt to center the object in the frame (functions
down below)
    panTrack(x);
    tiltTrack(y);

    //tell the PIC that an object is located
    digitalWrite(seeing, HIGH);

    //read the values from the PIC to find follow distance adjustment
```

```
    distAdj = get_distance() + 1;
    Serial.println(distAdj);

    ////////////////////////////////////
    ////MOTOR DRIVE ALGORITHM START////
    ////////////////////////////////////

    //takes object hight detirmes forward drive speed from that
adjusted by PIC distance adjustment value
    //Speed forward is set to a maxium of 50% power without turning
    if (height < (dist - (pow(distAdj, 1.7))))
    {
      rSpeed = (dist - (pow(distAdj, 1.7))) - height;
      rSpeed = rSpeed * (50 / (dist - (pow(distAdj, 1.7))));
    }
    else
    {
      rSpeed = 0;
    }
    //set left speed equal to right speed to start with
    lSpeed = rSpeed;

    //map turn value from -150 to 150 from the value of the pan
position of the servo
    turn = map(panPos, 25, 185, -150, 150);

    //take turn value and adjust left and right motors to steer robot
    lSpeed = lSpeed + turn;
    rSpeed = rSpeed - turn;

    //if motor speed is calculated below 15% power just don't drive
motors
    //this helps prevent oscillation at low values and the robot won't
move at less than 5% power anyways
    if (rSpeed < 5 && rSpeed > -5) rSpeed = 0;
    if (lSpeed < 5 && lSpeed > -5) lSpeed = 0;

    ////////////////////////////////////
    ////MOTOR DRIVE ALGORITHM END////
    ////////////////////////////////////

    //read value from the pic to determine if we're in drive or hold
mode
    if (digitalRead(rMode) == HIGH) //DRIVE MODE
    {
      DriveL(lSpeed);
      DriveR(rSpeed);
      Serial.print("L: "); Serial.print(lSpeed);
      Serial.print(" R: "); Serial.println(rSpeed);
    }
    else //HOLD MODE
    {
      Serial.println("STAHP!");
```

```
      DriveL(0);
      DriveR(0);
    }

    //tally's amount of time passed between block sightings
    lastBlockTime = millis();
  }
  //if a too much time has passed since pixy saw an object
  else if (millis() - lastBlockTime > 100)
  {
    Serial.println("halp...");

    //tell the PIC there is no object to be seen
    digitalWrite(seeing, LOW);

    //STOP MOTORS
    DriveL(0);
    DriveR(0);
  }
}


////////////END OF PRIMARY DRIVE LOOP////////////
////////////END OF PRIMARY DRIVE LOOP////////////
////////////END OF PRIMARY DRIVE LOOP////////////
////////////END OF PRIMARY DRIVE LOOP////////////


//function to drive left motor, pass value between -100 and 100 to
//drive it, -100 being full speed in reverse
//100 being full speed forward
void DriveL(int pwrL)
{
  if (pwrL > 100) pwrL = 100;
  if (pwrL < -100) pwrL = -100;

  pwrL = map(pwrL, -100, 100, -255, 255);

  if (pwrL > 0)
  {
    digitalWrite(dirL, HIGH);
    analogWrite(powerL, pwrL);
  }
  else if (pwrL < 0)
  {
    pwrL = pwrL * -1;
    digitalWrite(dirL, LOW);
    analogWrite(powerL, pwrL);
  }
  else
  {
    analogWrite(powerL, 0);
  }
```

```
}

//fucntion to drive right motor, pass value between -100 and 100 to
//drive it, -100 being full speed in reverse
//100 being full speed forward, right motor is inverted from left
//motor, function still behaves the same but
//high and low outputs on direction pins are swapped because forward
//for the right motor is backwards in relation to the robot's body
void DriveR(int pwrR)
{
  if (pwrR > 100) pwrR = 100;
  if (pwrR < -100) pwrR = -100;

  pwrR = map(pwrR, -100, 100, -255, 255);

  if (pwrR > 0)
  {
    digitalWrite(dirR, LOW);
    analogWrite(powerR, pwrR);
  }
  else if (pwrR < 0)
  {
    pwrR = pwrR * -1;
    digitalWrite(dirR, HIGH);
    analogWrite(powerR, pwrR);
  }
  else
  {
    analogWrite(powerR, 0);
  }
}

//pans the pixy left and right to keep track of the object
void panTrack(int x)
{
  panSpeed = x - 160;
  panSpeed = panSpeed / 16;
  //if pan is already at max or min position, set position to max or
min position
  if (panPos >= 186)
  {
    panPos = 185;
  }
  else if (panPos <= 26)
  {
    panPos = 27;
  }
  else //change pan position by increment detirmined by distance of
object from center
  {
    panPos = panPos + panSpeed;
  }
  setPan(panPos);
```

```
}
//tilts the pixy up and down to keep track of the object
void tiltTrack(int y)
{
  tiltSpeed = y - 100;
  tiltSpeed = tiltSpeed / 12;
  if (tiltPos >= 180)
  {
    tiltPos = 179;
  }
  else if (tiltPos <= 50)
  {
    tiltPos = 51;
  }
  else //change tilt position by increment determined by distance of
object from center
  {
    tiltPos = tiltPos + tiltSpeed;
  }
  setTilt(tiltPos);
}
//function to set pan servo position by mapping degrees to duty cycle
pulse rate
void setPan(int panPos)
{
  //panPulse = map(panPos, 26, 185, PANMIN, PANMAX);
  //pwm.setPWM(pan, 0, panPulse);
  pan.write(panPos);

}
//function to set tilt position by mapping degrees to duty cycle pulse
rate
void setTilt(int tiltPos)
{
  //tiltPulse = map(tiltPos, 50, 180, TILTMIN, TILTMAX);
  //pwm.setPWM(tilt, 0, tiltPulse);
  tilt.write(tiltPos);
}

//function that returns a value from 0 to 7 from based off combination
of inputs to A0 through A2
byte get_distance(){
  byte dist;

  //assign value to distance using pins as 3 least significant bits
  dist = digitalRead(distIn0) | digitalRead(distIn1) << 1 |
digitalRead(distIn2) << 2;

  return dist;
}
```

PIC:

```
'*****************************************************************
'*  Name    : PIC_CODE.BAS                                       *
'*  Author  : GROUP 42                                           *
'*  Notice  : Copyright (c) 2016 [select VIEW...EDITOR OPTIONS] *
'*          : All Rights Reserved                                *
'*  Date    : 12/9/2016                                          *
'*  Version : 1.6                                                *
'*  Notes   :                                                    *
'*          :                                                    *
'*****************************************************************

DEFINE OSC 8
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1

TRISA = %00000000
TRISB = %10001111

ANSEL = 0

'input pins from rf receiver
hold          Var    PORTB.0
drive         Var    PORTB.1
farther       Var    PORTB.2
closer        Var    PORTB.3

'input pin from Arduino
seeing        Var    PORTB.7

distance      var    byte

'output pins to LED
redpin        var    PORTA.1
greenpin      var    PORTA.2
bluepin       var    PORTA.3

'output pins to Arduino
running       Var    PORTA.0
dist_0        var    PORTB.4
dist_1        var    PORTB.5
dist_2        var    PORTB.6

distance = 4   'set initial distance value to middle of range
low running    'start in hold state (motors don't run)
```

```
init          var    byte   'initialization counter variable
i             var    byte   'object lost/scan counter variable

low bluepin
low redpin

'initialization sequence
'blink red LED twice per second for 3 seconds
init = 0
For init = 0 to 6
  high redpin
  pause 250
  low redpin
  pause 250
Next init


'main loop
Do

  'use state of rf receiver pins
  'to set running mode pin

  'running mode turned off when hold button pressed
  'and turned on when drive button pressed
  if (hold) then
     low running
  elseif (drive) then
     if (seeing) then
        high running
     else
        gosub Scan_from_stopped
     endif
  endif

  'handle increment/decrement rf pins
  if (farther) then
     if (distance < 7) then
        distance = distance + 1
        pause 250
     endif
  endif
  if (closer) then
     if (distance > 0) then
        distance = distance - 1
```

```
        pause 250
      endif
    endif

    'execute scan subroutine if object is lost
    if (running AND NOT seeing) then
      Gosub Scan_lost
    endif

    'set led pins
    if (running) then     'LED is green when in running mode
      low redpin
      high greenpin
      low bluepin
    elseif (seeing) then  'LED is blue when holding and object detected
      low redpin
      low greenpin
      high bluepin
    else                  'LED is red when holding and no object detected
      high redpin
      low greenpin
      low bluepin
    endif

    'assign 3 lsb of distance to pins
    dist_0 = distance.0
    dist_1 = distance.1
    dist_2 = distance.2

  loop


  Scan_lost:
    low redpin
    low bluepin
    For i = 1 to 20
      low greenpin
      pause 100
      high greenpin
      pause 100
      if (seeing) then  'exits to main loop if object is found
        Return
      endif
      if (hold) then       'checks for hold button in this subroutine
        low running      'as well as main loop
        Return           'due to pauses in this subroutine
```

```
    endif
  Next i
  low running          'running mode off if object lost for 4 seconds
  Return

Scan_from_stopped:
  low greenpin
  low bluepin
  For i = 1 to 20
    low redpin
    pause 100
    high redpin
    pause 100
    if (seeing) then  'runs and exits to main loop if object is found
      high running
      Return
    endif
    if (hold) then       'checks for hold button in this subroutine
                  'as well as main loop
      Return          'due to pauses in this subroutine
    endif
  Next i
  low running          'running mode off if object lost for 4 seconds
  Return
```