engineering science 50, spring 2013

RC HOVERCRAFT

a final project presentation
by Ishan Chatterjee,
Jenny Chang, and Daniel Yue
Instructors: Marko Loncar
and Evelyn Hu

School of Engineering and Applied Sciences
Harvard University
33 Oxford Street
Cambridge MA.

RC HOVERCRAFT

Ishan Chatterjee, Jenny Chang, and
Daniel Yue, May 6, 2013

contents

RC HOVERCRAFT

Ishan Chatterjee, Jenny Chang, and
Daniel Yue, May 6, 2013

abstract

Our goal entering this project was to make something tangible and cool, and so we decided to try to make a device that did something physical. The idea of a hovercraft appealed to us because it was something unusual and outside of the realm of devices that people normally make (boe-bots- been there, done that). Yet it also presented an opportunity to learn about mechanical design and control. Also appealing to us was the idea of wirelessly controlling something that moved, and the hovercraft fulfilled this role well.

introduction

The overall goal of this project was to have a functioning remote-controlled hovercraft. The specifics of its operation are expanded upon in the design section, but basically the final hovercraft comprises one acrylic plate, two fans, and one servo. One fan is used to lift the craft, one is used to power it forward, and the servo controls a wheel that steers from behind. Two arduinos allow for remote control, with xbee radio devices being used for wireless communication. The arduino attached to the hovercraft itself was responsible for interpreting the commands that we sent from our remote control. Ultimately, the lifting of the device comes from the inflation of a plastic skirt that sits under the craft, allowing for it to ride on a pocket of air and eliminate friction with the ground. We chose this project because we were interested in the ideas of remote control and mechanical design. This is cool because... well, it's a hovercraft, and it can be controlled wirelessly!

Ishan Chatterjee, Jenny Chang, and
Daniel Yue, May 6, 2013

design

general design

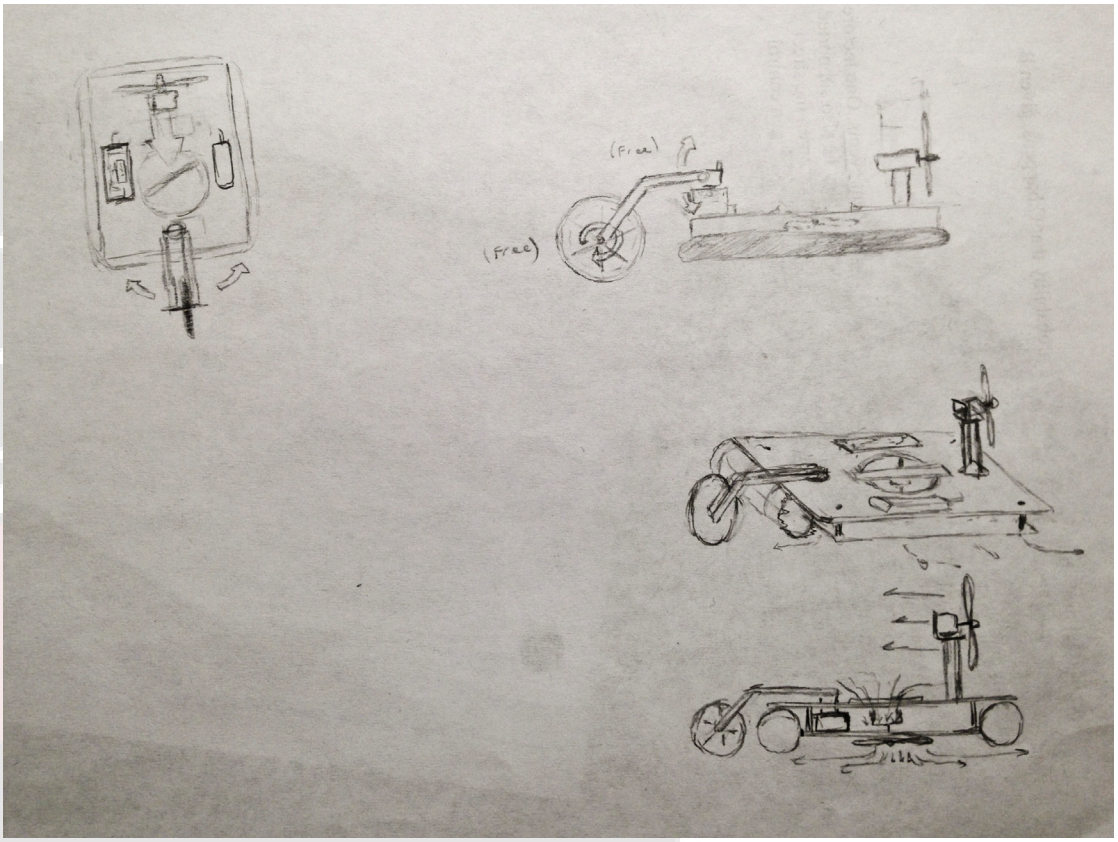
Lift: Lift is created when a downward facing propeller powered by a motor inflates the plastic skirt, creating a pocket of air and pushing air out from underneath the skirt. This lifts the craft and reduces friction, allowing the craft to hover off the ground.
Drive: The craft is driven by an outward facing propeller powered by a motor.
Steering: Steering is completed by a horizontally rotating lever arm powered by a rotational servo motor. Attached to the end of the lever arm is a freely rotating arm with a free-spinning wheel as the point of friction with the ground.
Chassis: The main body of the craft is a single lightweight acrylic plate to which all of the mechanical and electrical components mount.
Control: joystick pad, Electronic Speed Controllers (ESCs), and XBees.

design process

We started by deciding on the main components that would be necessary to implement a hovercraft and choosing the simplest solutions to reduce complications, allow for easier troubleshooting, and lower costs. We thought that the simplest way to lift and drive the craft would be to have a single propeller and motor (regulated by an ESC) for each of the two tasks. The next necessary component was a steering mechanism. We considered several steering options, including additional propellers pointing in different directions, but ultimately decided that a single non-continuous rotation servo motor attached to a lever arm and free-spinning wheel would be the easiest to build and control. Our initial chassis design consisted of two plates separated by spacers. All the electrical components, steering mechanism, and drive mechanism attached to the top plate. The lift fan attached to the top plate and fit through a hole in the baseplate. We chose to make the chassis plates out of 1/8th inch thick acrylic to ensure that the chassis was robust yet lightweight. However, in the end, we realised that we only needed one plate for the hovercraft to function, which allowed us to lessen the weight that the fan needed to lift.

Our initial idea for remote control was to utilise a WiiMote controller. However, we quickly realised that the WiiMote would not work for two reasons- first, it's coding would be rather complicated and nobody in our group has extensive coding experience, and second, it's range was limited to 5 meters, which would not be optimal for function. We next considered a PlayStation 2 remote, which could similarly hooked up

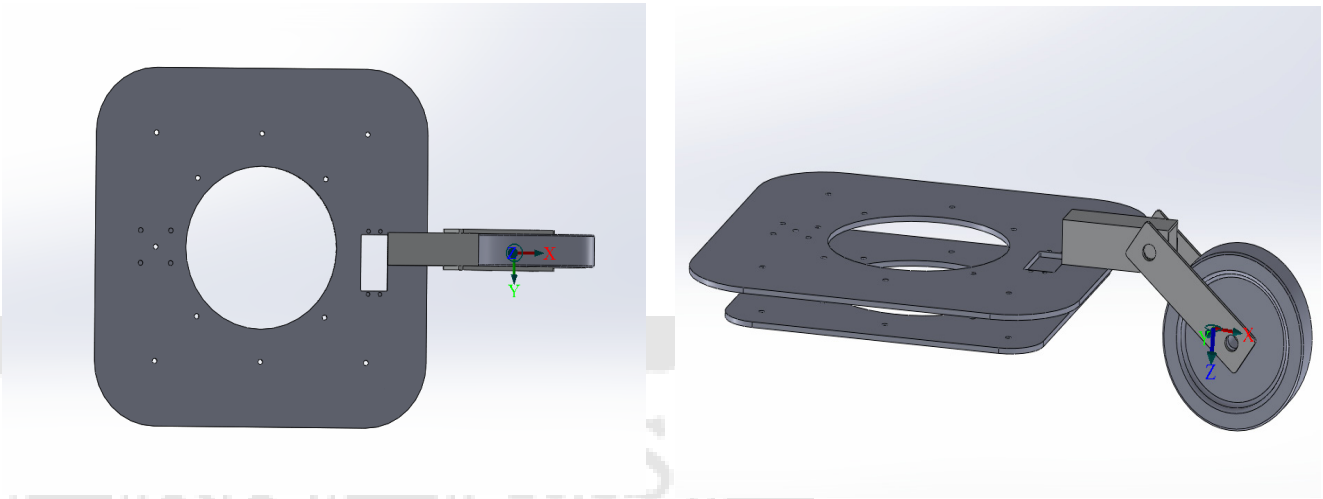
to an Arduino-XBee combination, achieving the same desired wireless control. However, when surfing sparkfun.com, we realised that there was a simple joystick package that would make for easy coding, and so we ultimately elected to use that in combination with xbee and arduino to achieve wireless control.



These are our initial plans for the design of the hovercraft. The basic elements of design remained the same throughout the process of its creation and evolution.

solidworks

Once we had several detailed sketches of the design, we made all of the parts on the computer-aided design program SolidWorks. From there, we were able to make an assembly of all the parts to visualize the 3D design and make sure all the components fit together well. (Sketches on the following page.)



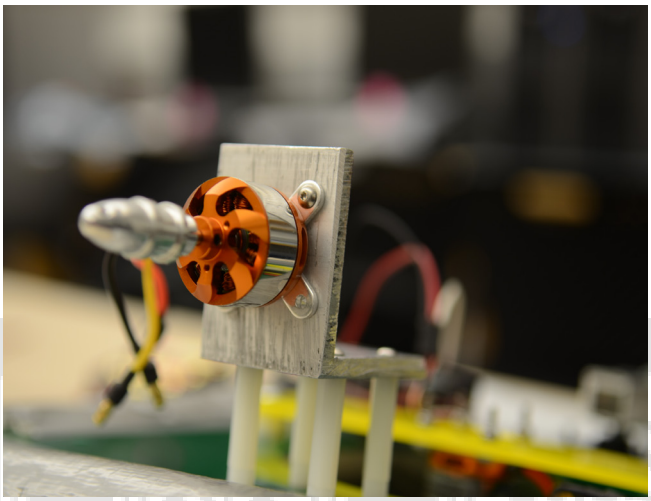
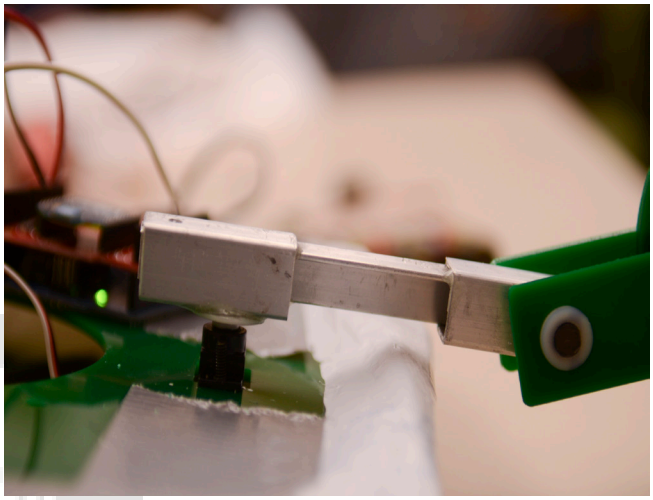
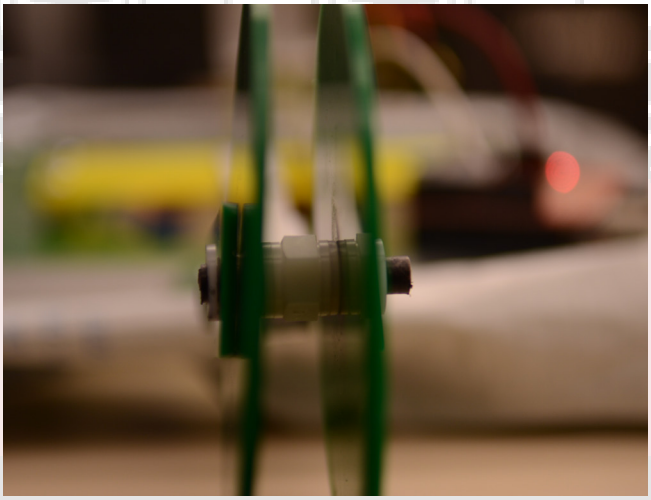
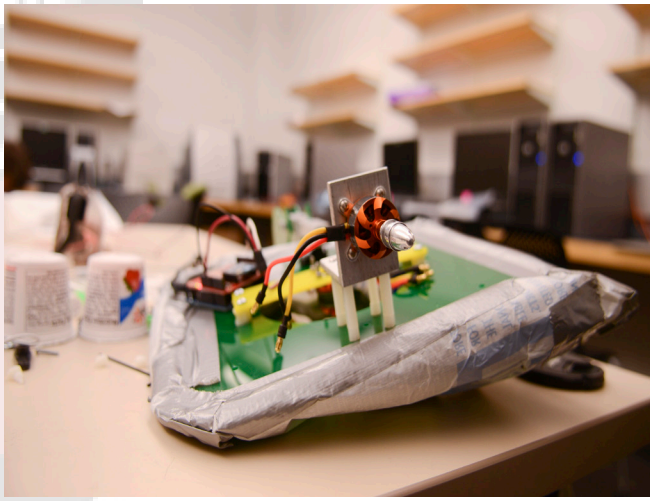
These are our CAD files, created in SolidWorks, which allowed us to laser cut the chassis of the craft. Note the bottom plate of which we eventually decided to rid ourselves.

fabrication

For the acrylic parts (chassis plates, steering wheel, lift motor mount), we exported the SolidWorks files to vector files (.dxf format) and cut them out on the laser-cutter. The remainder of the parts (drive motor mount, servo motor mount, lever arm) were made out of aluminum for strength, and were machined using the vertical band-saw, horizontal band-saw, and drill press.

assembly

Once the parts were machined, we began the assembly of the craft. For the most part, the components fit together well. We came across a few problems because the plastic screws we wanted to use kept breaking, so we switched to using metal screws to eliminate this problem. (Images on the following page.)

These are images of the various components of our hovercraft. Upper Left: the front drive fan. Upper Right: the connection of the steering arm attached to the servo. Lower Left: the wheel device, held together by a metal rod and spacers. Lower Right: The whole hovercraft, before the electronic devices were loaded. Note the skirt and the overall shape,

testing

After testing the craft, we decided to remove the bottom plate entirely because it was unnecessary and extra weight. We also initially ordered 5 inch diameter propellers, but when they arrived we thought they looked too flimsy, so instead we used 8 inch propellers that gave us a lot more power and lift. Additionally, the lift motor sat too

low and hit the ground, so we added spacers to the motor mount to raise the lift motor assembly higher with respect to the chassis.

We came across several issues when it came to the steering mechanism. We found that we ordered the incorrect servo motor (continuous rotation), which would be much more difficult to program and control. Thankfully, we found the correct servo in the ES 50 lab, but this new servo was different in size and thus we had to changed both the servo mount to the chassis and the lever arm mount to the servo motor. Since the new servo motor was smaller than the original, we had to modify the chassis plate and re-cut the plate on the laser-cutter. We also had to design and fabricate a new mount for the lever arm to the servo. Once we put the new arm assembly together and attached it to the chassis, we found that the arm was too short so that the lever arm could not turn and the wheel could barely touch the ground. Our solution to this was cutting the lever arm in half and gluing in a piece of aluminum angle stock, extending the length of the lever arm. With this new servo and extended arm, we came across another problem because the attachment between the servo and the arm was not robust and kept coming unattached. We tried several different epoxies and glues, none of which worked very well because the surface area of the connector was very small. Another option was trying to use a screw, but the small surface area would also make that challenging.

We also went through several iterations of the skirt design. We started out with a skirt made of a long balloon wrapped around the bottom of the craft. We found that this did not lift the craft high enough off the ground, so we added a second balloon so that the skirt was two balloon widths high. The issue with the balloon skirt was that balloons were bulky and prone to popping, so we finally fashioned a skirt out of an airtight plastic bag and duct tape. Furthermore, it was hard to get an even flow of air under the balloon skirt because of the stiffness of the skirt.

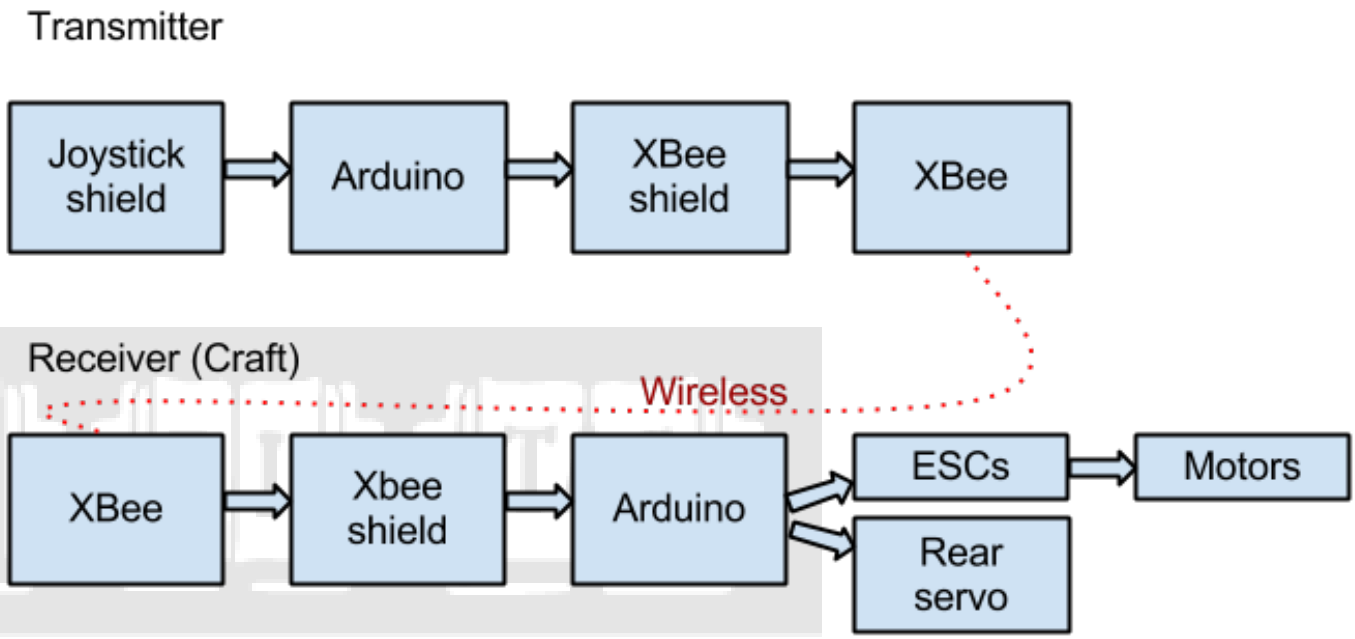
parts list

See **Link**:
https://docs.google.com/a/college.harvard.edu/spreadsheet/ccc?key=0AhYdhaK0V1g_dFFGckt3bTVseF85UUw4YjUtN1Y3SEE#gid=0
Or **Alternately**:
https://drive.google.com/a/college.harvard.edu/?tab=wo#folders/0BxYdhaK0V1g_UmFFcUt1eFNyT28
List is also on the next page for convenience.

Parts to Order	Supplier	URL	Quantity	Individual price	Price
Xbee Shield WRL-10854	sparkfun	https://www.sparkfun.com/products/10854	1	24.95	24.95
Turnigy Multistar 30 Amp Multi-rotor Brushless ESC 2-4S (OPTO)	hobby king	http://www.hobbyking.com/hobbyking/store/__27776__Turnigy_Multistar_30_Amp_Multi	2	10.29	20.58
Joystick Shield Kit DEV-09760	sparkfun	https://www.sparkfun.com/products/9760	1	12.95	12.95
ZIPPY Flightmax 2200mAh 3S1P 25C	hobby king	http://www.hobbyking.com/hobbyking/store/__7636__ZIPPY_Flightmax_2200mAh_3S1	1	8.99	8.99
Male XT60 connectors (5pcs/bag) GENUINE	hobby king	http://www.hobbyking.com/hobbyking/store/__10414__Male_XT60_connectors_5pcs_b	1	1.92	1.92
Hobbyking DC-4S Balance Charger & Cell Checker 30w 2s~4s	hobby king	http://www.hobbyking.com/hobbyking/store/__21044__Hobbyking_DC_4S_Balance_Ch	1	8.99	8.99
Amico 20 PCS Assorted Latex Long Balloons Party Favors	Amazon	http://www.amazon.com/Amico-Assorted-Latex-Balloons-Favors/dp/B008P1AGA0/ref=rec_dp_0	2	3.91	7.82
5030 Propellers (Black) - 3xCW and 3xCCW - 6pcs per bag	hobby king	http://www.hobbyking.com/hobbyking/store/__22753__5030_Propellers_Black_3xCW_a	1	3.49	3.49
(Motors) D2822/14 Brushless Outrunner 1450kv	hobby king	http://www.hobbyking.com/hobbyking/store/__12916__D2822_14_Brushless_Outrunner	2	8.31	16.62
Balloon pump	amazon	http://www.amazon.com/Balloon-Inflator-Double-Action-Pump/dp/B000TH5LL2/ref=sr_1_2?ie=UTF8&qid=1366074093&sr=8-2&keywords=cheap+balloon+pump	1	6.99	6.99
SHIPPING - Hobby King	hobby king		1		14.99
Parts we will buy					
Packing tape	Staples		1	3	3
				Total Cost	116.3
Plastic bag	Jenny	Jenny	2	Free	
Parts from ES50 Lab					
Plastic standoffs, screws, nuts, etc.					
Battery connectors for Zippy					
Acrylic					
Battery Pack for Arduino			1? 2?		
Wheel, axle, mounting arm					
Arduino Uno					
XBee 1mW U.FL Connection - Series 1 (802.15.4)					

project implementation

When we had decided upon a final design, we investigated what components we needed and how they worked. A great resource for us was the Quadcopter project from 2012, which not only involved motors, props, and ESCs but also XBee control. To tackle the project efficiently we split it into two main components, control and the craft, which we could pursue in parallel. The control aspect involved getting motors to respond to the joystick movements on the joystick pad. The ultimate control setup looked like this: (see next page).



At the heart of control is the wireless communication with the XBee modules. We initially thought this would be the most difficult part of the project. However, getting the XBees to communicate was in itself not difficult. Next, we tackled getting the joystick shield to communicate with the Arduino. Again, this was relatively straightforward as the joystick shield was created specifically for Arduino; the buttons fed directly into a digital in pin and the x and y position of the joystick could easily be read using analog in. The next part was to send these values read from the joystick and buttons over the XBee serial interface and read them on the other side. This was perhaps the most painful part of the entire project. We made the assumption that the serial interface could be read like a book -- that is if we transmitted 1 2 3 via XBee that the receiver would exactly receive 1 2 3 on the other end. While this assumption was not entirely incorrect, we found it didn't hold up when transmitting values at a baud rate of 9600. Instead the receiver sometimes read values twice or not at all; for instance, the received message might have read 2 3 3. This is an issue as we sent data for x joystick position, y joystick position, and button press data in parallel. If we just sent values without specifying which values corresponded to which controls, how would the receiver know which of these controls the data represented?

To overcome this issue, we found code online that used several different methods. The code we found either used the idea of "handshakes," periodic communications between the Xbees between sending packets of actual data to ensure that both were timed together, or "tagging," attaching a character tag for instance to pieces of data so

that the transmitter knows exactly what piece of information it is receiving (for example: http://ams-design.com/arduino/2011/09/the-spideruinos-controller/). We tried both of these methods, however had significant issues in parsing the data that was sent. Much of this was due to problems in data type communication. For example, different data types (int, char, string, byte) all had a different number of bytes that had to be sent. Handshaking required many complex exchanges just to get one piece of information from the controller to the receiver. We came up with a simple algorithm of our own, leaving all the baggage that came with two-way communication and data type issues behind. Every piece of information was sent as a single byte, a value from 0 - 255. We mapped the button for lift fan to the values 0 and 1, the x joystick values to 2 - 128, and the y joystick values from 129 - 255. In this way, in one byte the data sent contained information on the control value and which control it came from. The loss in acuity for the analog values was inconsequential as we still had a 126 value gradient with which to operate.

As our motors and ESCs (electronic speed controls) had yet to arrive, we hooked the Arduino receiver up to LEDs on a breadboard to model the motors. Later on, once the ESCs and motors arrived we added the necessary code to interface with the ESCs. We used the servo library, declaring our motors as servos, initializing the ESCs by sending a zero PWM signal (servo.writeMicroseconds(1000)), and using the write-Microseconds command to modulate the speed. For the rear servo we used the write command to input an angle. The left button when pressed toggled the lift fan on and off. The x position of the joystick turned the back servo left and right and the y position of the joystick changed the speed of the drive fan from zero to max.

RECEIVER CODE

```
#include <Servo.h>     // Include servo library

byte  incomingByte;
int incomingInt;
int button_value;
int button_value_prev;
int joystick_x_value;
int joystick_y_value;

Servo   servoSteer;// Declare servos
Servo servoLift;
Servo servoDrive;

int lift_fan = 1;
int steerAngle = 90;
int driveSpeed = 1000;

void setup() {
  Serial.begin(9600);      // open the serial port at 9600 bps
  servoSteer.attach(9);
  servoLift.attach(6);
```

```
  servoDrive.attach(11);
  servoLift.writeMicroseconds(1000); // Send zero PWM signal to intialize ESCs
  servoDrive.writeMicroseconds(1000);
  delay(1000);


}

void loop() {

  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
     incomingByte =Serial.read();
     incomingInt =int(incomingByte);

    //Determine what kind of input the value represents
    if    (incomingInt      <=      1)      {
      button_value_prev = button_value;
      button_value = incomingInt;
    }

    if (incomingInt > 1 && incomingInt <= 128) {
      joystick_x_value = incomingInt - 2;
    }

    if (incomingInt > 128) {
      joystick_y_value = incomingInt - 129;
    }

    Serial.println(button_value); // print values to serial monitor
    Serial.println(joystick_x_value);
    Serial.println(joystick_y_value);

  }

   // logic to toggle lift fan on and off
   if (button_value == 0 && button_value_prev != button_value) {
          if (lift_fan == 0){
            lift_fan = 1;
          }
          else if (lift_fan == 1){
            lift_fan = 0;
          }
        }


        if (lift_fan == 0)
        {
          servoLift.writeMicroseconds(2000);

        }
```

```
        else {
          servoLift.writeMicroseconds(1000);
        }

      // map joystick y values to drive motor speed
       joystick_y_value =constrain(joystick_y_value,  63,  126);
       driveSpeed =map(joystick_y_value, 63, 126, 1000, 2000);
      servoDrive.writeMicroseconds(driveSpeed);

      // map joystick x values to steer servo angle
       steerAngle =map(joystick_x_value, 0, 126, 141, 51);
      servoSteer.write(steerAngle);

  }
```

TRANSMITTER CODE

```
// Based on code  written for SparkFun Arduino Inventor's Kit CIRC-JOY

// Store the Arduino pin associated with each input
const byte PIN_BUTTON_SELECT = 2;
const byte PIN_BUTTON_RIGHT = 3;
const byte PIN_BUTTON_UP = 4;
const byte PIN_BUTTON_DOWN = 5;
const byte PIN_BUTTON_LEFT = 6;

const byte PIN_ANALOG_X = 0;
const byte PIN_ANALOG_Y = 1;

byte button_value;
byte joystick_x_value;
byte joystick_y_value;

void setup() {
  Serial.begin(9600);

  // Specify each pin connected to a pushbutton as an input.
  // Also enable the Arduino's internal "pull-up" resistors
  // for each pushbutton we want to read--this means the shield
  // doesn't need to have resistors on it.
  // Note that when a pull-up resistor is used on a pin the
  // meaning of the values read are reversed compared to their
  // usual meanings:
  //    * HIGH = the button is not pressed
  //     * LOW = the button is pressed
  pinMode(PIN_BUTTON_RIGHT, INPUT);
  digitalWrite(PIN_BUTTON_RIGHT, HIGH);

  pinMode(PIN_BUTTON_LEFT, INPUT);
  digitalWrite(PIN_BUTTON_LEFT, HIGH);

  pinMode(PIN_BUTTON_UP, INPUT);
  digitalWrite(PIN_BUTTON_UP, HIGH);

  pinMode(PIN_BUTTON_DOWN, INPUT);
  digitalWrite(PIN_BUTTON_DOWN, HIGH);

  pinMode(PIN_BUTTON_SELECT, INPUT);
  digitalWrite(PIN_BUTTON_SELECT, HIGH);

  //Delay startup by a little
  delay(1000);

}


void loop() {
```

```
  // read values and parse to byte
   button_value =byte(digitalRead(PIN_BUTTON_LEFT));

   joystick_x_value =byte(map(analogRead(PIN_ANALOG_X), 0, 1023, 2, 128)),

   joystick_y_value =byte(map(analogRead(PIN_ANALOG_Y), 0, 1023, 129, 255));

  // write to serial
  Serial.write(button_value);
  Serial.write(joystick_x_value);
  Serial.write(joystick_y_value);

}
```

In parallel, we worked on the mechanical design of the craft. Much of this portion is covered in the design section. First we created SolidWorks models of the hovercraft chassis and the arm components. We used a laser-cutter to produce many of the components. Using scrap aluminum from the lab we made the rear wheel assembly (see "design" for challenges with this component). We laser-cut an acrylic assembly to hold our lift fan motor as well. A motor mount was machined out of aluminum angle stock to hold the drive fan motor. These laser-cut and machined components interfaced with the motor via the motor mount that came with the part. Also the 8" propellers connected onto the motor shaft with a part that came with the motor.

As mentioned in the design section, the skirt originally started as a stack of long balloons wrapped around the chassis. There were multiple problems with this design, one of which was that the inflated balloons were not compliant enough to allow for an even flow of air out from beneath the pocket of positive pressure created under the craft. We therefore created a skirt out of a plastic bag and duct tape basing our design off of a design seen in a YouTube video (http://www.youtube.com/watch?v=A_cft-vNTxKA). We experimented with different versions of this skirt, mostly focusing on modifying the diameter of the central air outlet -- too wide of an air outlet and the skirt would not inflate, too narrow and lift would be sacrificed. We finally settled on an air outlet approximately 7.5 inches in diameter. We reinforced the skirt with duct tape to add strength while maintaining a light weight. A smaller servo and this new skirt design also allowed us to eliminate the second, smaller bottom plate entirely and have the pocket of air sit under the first plate. With some slight modifications to raise the lift fan, this new design shed weight and simplified the design.

Before the craft was fully built, we played around with the electrical components.

Rear servo:

Our original servo was a continuous rotation that we had to swap out for a normal servo for ease of programming (the servo.write() command differs based on the type of servo). This was very easy to hook up to the Arduino -- 5V, ground, and one signal wire.

ESCs and motor:

ESCs are usually connected to radio receiver modules for hobby RC planes, but still interfaced relatively easily with the Arduino (or at least we thought so until the very last couple hours before the design fair. . .) other than a couple programming quirks. The positive and negative leads connected to the battery. The one signal wire was connected to an Arduino pin. The signal ground was connected to the battery ground. We didn't use the 5V out. The three motor leads connected to the motor's input wires -- switching any two wires would cause the motor to spin the other way.

Motor:

Connected the three motor leads to the ESCs. Not much to note.

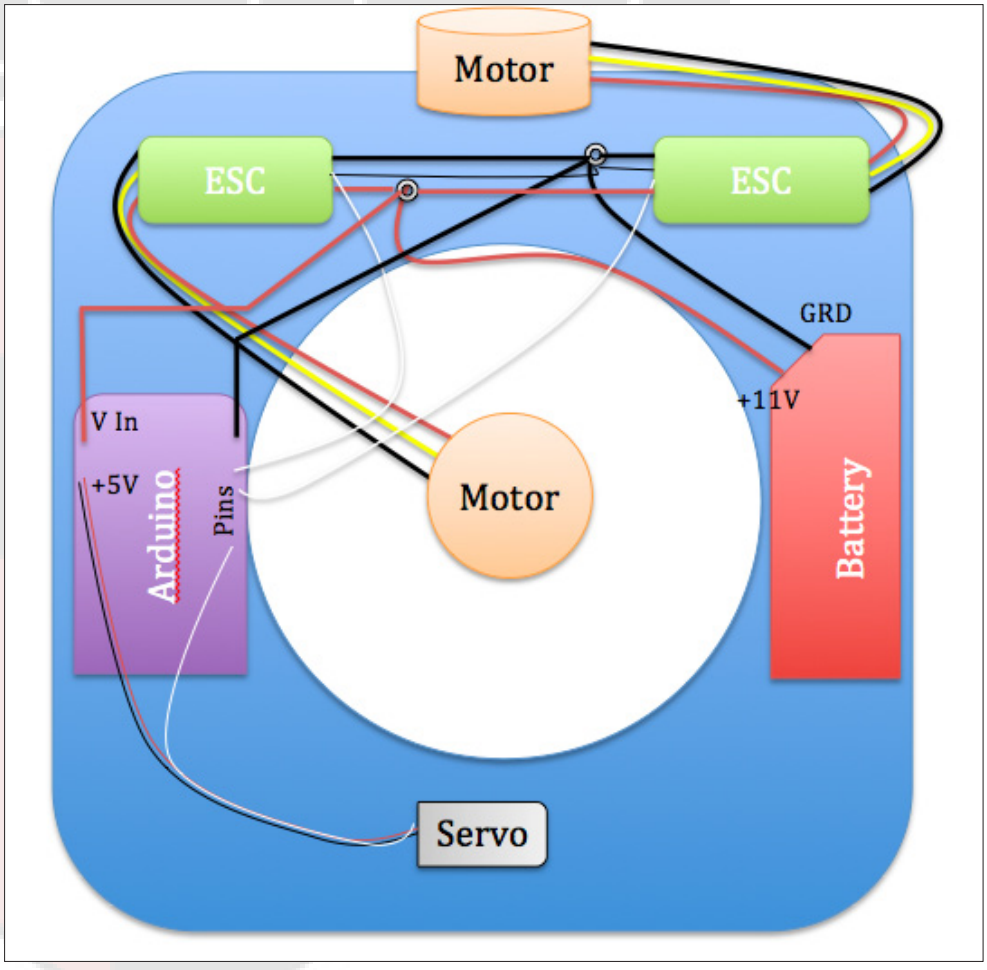
Battery:

The 11.1V 3 cell battery was connected in parallel with the two ESCs and the Arduino. Two metal clips serve as the physical nodes for the point at which all the components are connected.

Arduino:

Powered directly off of the 11.1V battery (plugged into V In and ground). The 5V out from the Arduino serves to power the rear servo. The Xbee shield and Xbee sit atop the Arduino and are powered directly from it. The signal leads from the ESCs and the rear servo are plugged into PWM output pins.

A wiring diagram is provided below:



As shown above the ESCs and Arduino board are hooked up in parallel to the battery such they all have 11.1V across their leads. Not included in the above diagram that between the battery and the leads there is a battery connector that allows for easy removal of the battery for powering on and off and charging. Once we connected everything properly, we cut away excess wire and neatened and taped down the wires. The components were attached to the acrylic chassis via double-sided, adhesive foam.

team management

Our goal entering the project was to split work relatively evenly, with all of us working on the components together. Ultimately, due to time constraints and availability, this did not ultimately end up happening. We still worked on most aspects of together. However, Jenny ultimately did the lion-share of the machining and mechanical work, because of her prior experience working on robots and knowledge of SolidWorks. Ishan and Daniel worked more on the electrical aspects of the project, including the remote control, the motors, and the arduinos. However, everybody was responsible for coming up with solutions to the various issues that arose, and ultimately this project was really a product of all of our work.

outlook

Obviously the biggest issue is that we never got all the components working at one time. Although we did extensive troubleshooting of the motors, we couldn't figure out why they were not working. With more time, we definitely would go back to square 1 and figure out why it wasn't working. If we could get another ESC, we could hook it up and see if broken ESCs were our issue, or if there is some more fundamental problem happening underneath. Besides that, I think we could develop a more effective way of connecting the back lever arm to the servo- we had connect issues due to the smaller rotation nut on top, where the wheel would simply torque itself off due to its weight and off-center placement (it formed a really long lever arm).

acknowledgements

Many people helped us with our project, but we'd like to take the time to recognize a few outstandingly helpful people: Ben Dickensheets for meeting with us, talking over design, helping us set up our system, and troubleshooting. Also, thanks for helping us understand how xbees work; Ellen Farber, for showing us the ropes on the laser cutter and machine shop, and also talking through mechanical design issues. Also, thanks to Marko for his dedication, help finding a non-continuous rotation servo, and morale support (and pizza!) in the final stretch. Lastly, thanks to Xuan for keeping the ES50 lab in order, and for ordering our parts.

disclaimer

It is perfectly okay to share our report and code with other people.

references

Electronic Speed Controller manual- http://www.hobbyking.com/hobbyking/store/uploads/809043064X351363X29.pdfArduino Servo Library- http://arduino.cc/en/Reference/Servo
Hovercraft tutorial- Part 1: http://www.youtube.com/watch?v=A_cftvNTxKA
This one is awesome!!!!!- Part 2:
http://www.youtube.com/watch?feature=fvwp&NR=1&v=jsFVhGlATAA
Spideruino's controller- http://ams-design.com/arduino/2011/09/the-spideruinos-controller/
Quadcopter Project from ES50 2012