

The only hand tools you will need are a small phillips-head screwdriver, a 1/4" wrench, and a 1/8" drill bit/drill, but you will need access to a laser cutter to cut the acrylic. I used a laser cutter at TechShop in Menlo Park. TechShop is a DIY workshop, if you are interested check out their website, www.techshop.ws.



Cut the acrylic

The first step is to cut the acrylic. I have attached the files I used, which you can also modify if you want. Make sure to check fitment on the holes for the servo mounting, etc. to make sure the fitment will be right before cutting everything out only to find the holes are too loose.

I created these files using Autodesk's AutoCAD Mechanical 2012, which is also available at TechShop if you don't have a copy of your own.

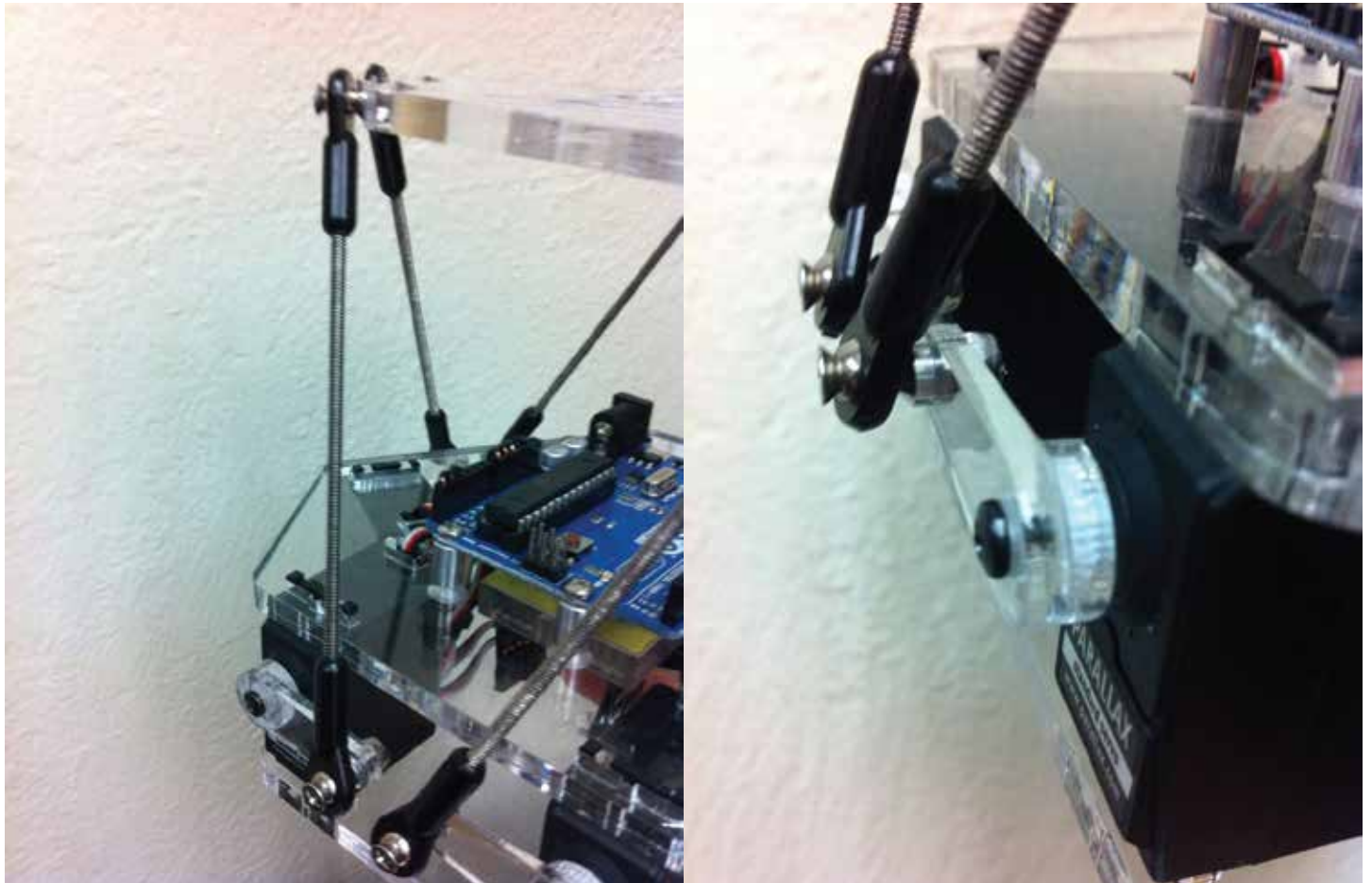
There are 3 files to be cut out of 1/4" acrylic: `base_bottom.eps`, `base_top.eps`, and `platform.eps`

And 1 file to be cut out of 1/8" acrylic: `servo_arms.eps`

Assemble the connecting links

The next step is to cut the rods to length. I used 2x 1 ft. sections, which I cut into 6x 4" pieces. Make sure that the ends are cleaned up nicely because they will be threading into plastic which can be easily damaged by the metal if there is a burr.

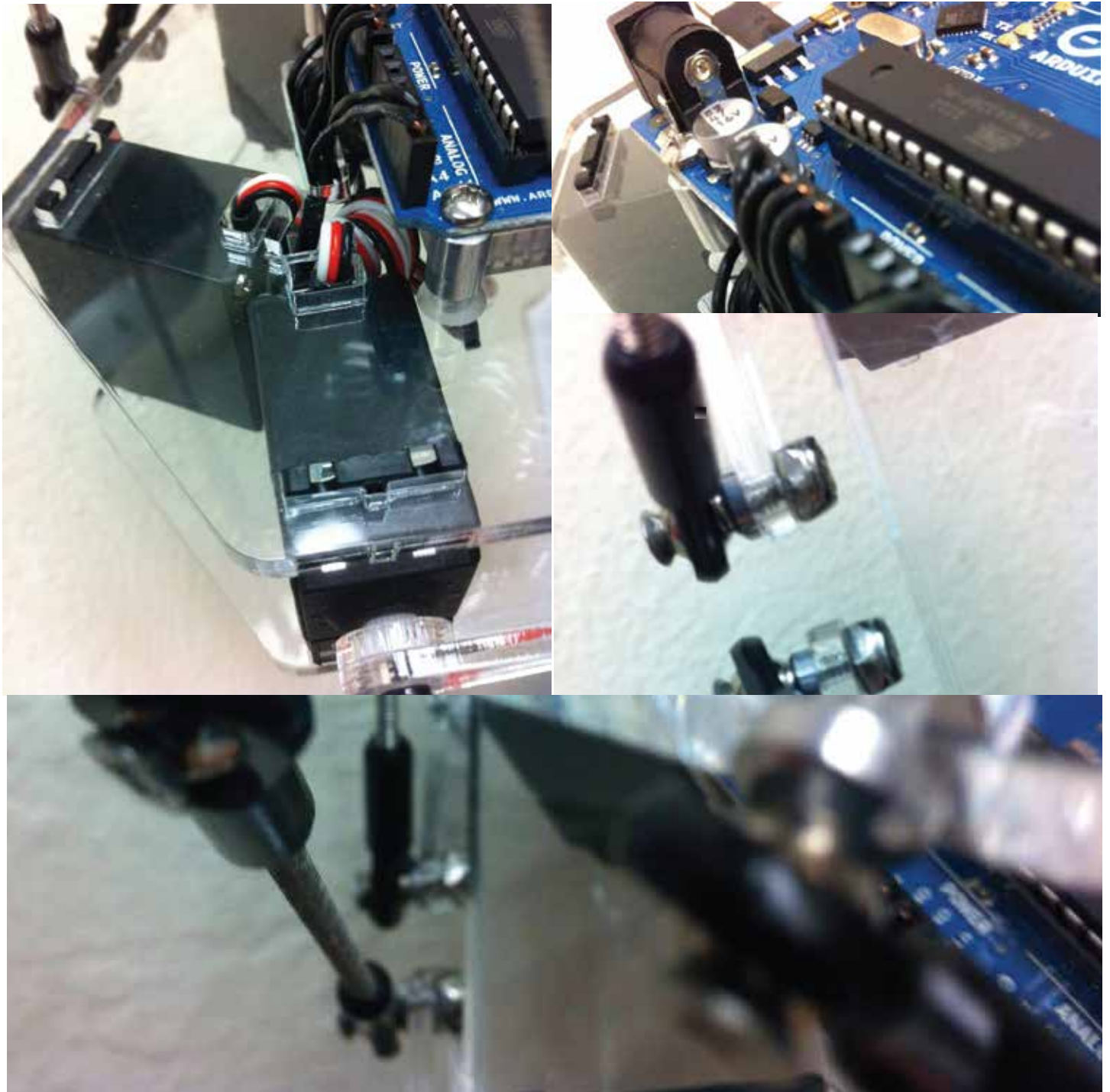
Once the rods are cut to length, simply thread a rod end onto each end of each rod, making sure that all 6 rods are exactly the same length from end to end when you are done.



Glue and install the servo arms

The next step is to glue and install the servo arms onto the servos. I used a liquid acrylic solvent from TAP plastics, but you can use any glue you like. With the liquid solvent approach, I found it helps to lightly sand the surfaces being glued to ensure they stay together.

Glue the circular piece to the arm as shown in the picture, making sure that the splined piece still fits onto the servo arm (the laser creates a slightly tapered cut, so the splined pattern will fit on the servo shaft in one orientation better than the other)



Assemble the platform

Start by threading the servo connectors through the base_top piece from the bottom up, then push the servo into place and thread the connector back down through the other hole. Do this for all 6 servos.

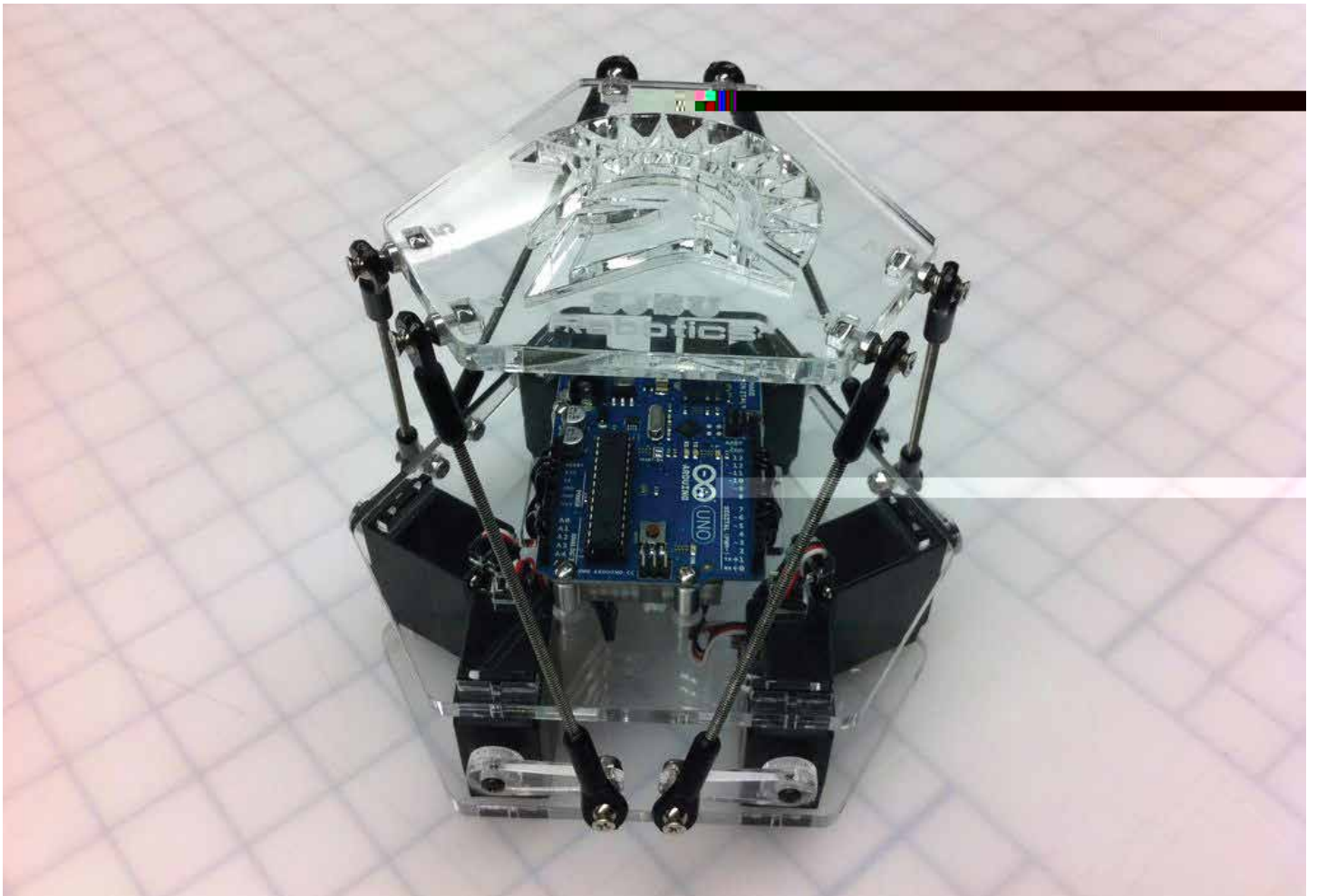
Now you can mount the Arduino using the threaded aluminum spacers, 3/16" pan head screws, and 11/16" flat head screws. To make it look nice you can countersink the bottom side of the holes in the acrylic so the flat heads sit flush, but it's entirely necessary. You may also need to shave off a bit of the aluminum spacers so that they fit flat against the Arduino (you'll see what I mean when you try to assemble them).

Once the Arduino is mounted, you can stick the breadboard right underneath it, in between all the servos. With the breadboard in place, you can wire everything up. The servos all need +5V on the red wire, ground on the black wire, and signal from the Arduino on the white wire. Make sure you use the PWM capable pins on the Arduino (I used pins 3, 5, 6, 9, 10, 11). Route the wires as shown in the picture.

Once everything is wired up, you can install the base_bottom piece. As you can see there is no mounting hardware for the servos, it's just a "press" fit. This is for ease of assembly/disassembly/debugging, so don't worry if you didn't get something wired up correctly.

Now it's time to add the connecting links. Start by attaching the links to the servo arms. Use a flat head screw through the rod end, then a conical spacer, then the arm, and into a nylok nut on the back. Before attaching the top, you'll need to drill 1/8" holes in the sides of the platform where the holes for the nuts are. Once you've drilled the holes, the connecting links attach the same way they did on the bottom: a flat head screw through the rod end, then a conical spacer, then the acrylic and into a nylok nut embedded in the platform. Be careful not to overtighten these as the plastic can easily crack.

You're done! You can play around with the programming yourself, or you can check out my other instructable on how to program the platform.



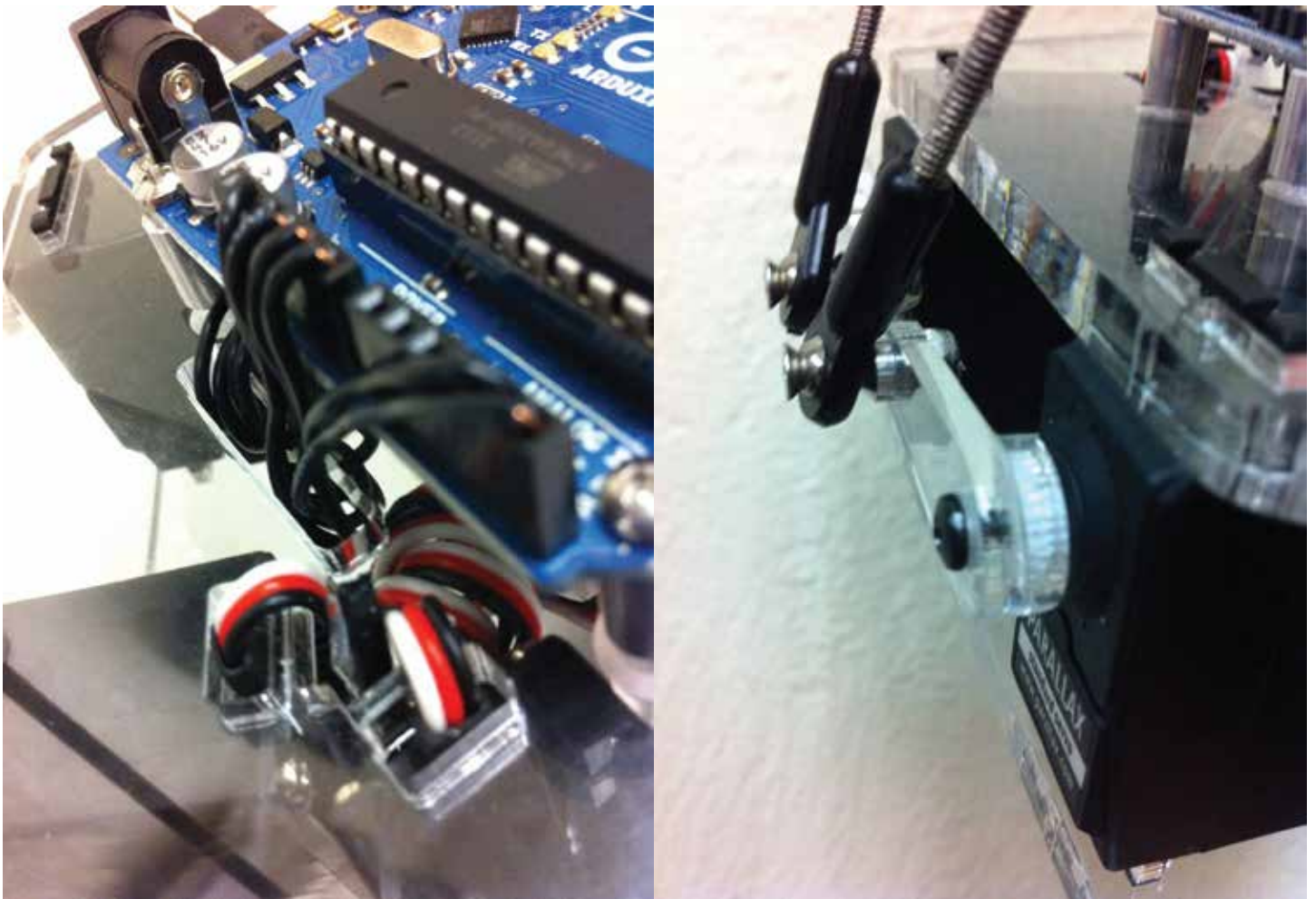
How to program an Arduino-powered 6DOF motion platform

This is a follow up to my previous instructable about building an Arduino powered 6 degree of freedom motion platform. In this instructable, I'll talk about programming the platform using an inverse kinematics approach.

I built the platform using a laser cutter at the TechShop in Menlo Park, CA, and I started learning about the Arduino microcontroller in TechShop's Arduino class. TechShop is a DIY workshop, if you're interested check out their website at www.TechShop.ws

In robotics, inverse kinematics, as opposed to forward or direct kinematics, is the process of determining the positions of each actuator necessary to achieve a given position of the end effector (in this case the platform), rather than determining the position of the end effector given the positions of each actuator.

I won't go into too much detail on the math, but the basic idea is this: The microcontroller calculates the position of the upper mounting points for the connecting links by taking into account the desired position and orientation of the platform relative to the base along with the position of the mounting points relative to the platform itself. It then calculates the current position of the lower mounting points based on the current position of the servo arm and its position relative to the base. The distance between these two points is then compared to the physical length of the connecting links and the servo arm is moved up or down accordingly.



Here is my code, I have commented it with descriptions of each variable, and I'll try to explain as much as I can here about how to use it. I have also attached the .ino file if you want to load it directly into the Arduino IDE.

Looking down at the platform with the USB port pointing away from you, the positive x axis is to your right, positive y away from you (in line with the USB port), and positive z is up. rotation follows the right-hand rule (point your thumb in the direction of the positive axis, your fingers curl in the positive rotation direction), so positive x rotation will tilt the platform back, positive y rotation will tilt the platform to the right, and positive z rotation will rotate the platform counter-clockwise.

The platform will "rest" at a specific height (z_home)

The array `pe[6]` contains the information about where the platform should be:

`pe[x position , y position, z position, x rotation , y rotation , z rotation]`

changing the x position, y position, and z position variables will move the platform accordingly (in inches). Changing the x rotation, y rotation, and z rotation variables will rotate the platform accordingly (in radians).

So tell the platform where you want it to be by changing the appropriate variables, then uploading the code. Once uploaded, the platform will initialize to the "home" position and pause for one second, then move to the desired position.

I'll talk about incorporating some type of realtime position input in another instructable. Alternatively, you could write a loop to run the platform through a series of movements such as this: <http://youtu.be/cfnBnx8Cvcg>

The code:

```
#include <Servo.h>
```

```
const float pi = 3.14159, theta_r = radians(48.0), theta_p = radians(23.2), theta_s[] = {-pi/3, 2*pi/3, pi, 0, pi/3, -2*pi/3},
```

```
RD = 2.395, PD = 3.3, L1 = 1.0, L2 = 4.72, z_home = 4.25, servo_min, servo_max, servo_mult
```

```
p[2][6] = {{PD*cos(pi/6 + theta_p), PD*cos(pi/6 - theta_p), PD*cos(-(pi/2 - theta_p)), -PD*cos(-(pi/2 - theta_p)), -PD*cos(pi/6 - theta_p), -PD*cos(pi/6 + theta_p)},
```

```
{PD*sin(pi/6 + theta_p), PD*sin(pi/6 - theta_p), PD*sin(-(pi/2 - theta_p)), PD*sin(-(pi/2 - theta_p)), PD*sin(pi/6 - theta_p), PD*sin(pi/6 + theta_p)}},
```

```
re[2][6] = {{RD*cos(pi/6 + theta_r), RD*cos(pi/6 - theta_r), RD*cos(-(pi/2 - theta_r)), -RD*cos(-(pi/2 - theta_r)), -RD*cos(pi/6 - theta_r), -RD*cos(pi/6 + theta_r)},
```

```
{RD*sin(pi/6 + theta_r), RD*sin(pi/6 - theta_r), RD*sin(-(pi/2 - theta_r)), RD*sin(-(pi/2 - theta_r)), RD*sin(pi/6 - theta_r), RD*sin(pi/6 + theta_r)}};
```

```
/*  
theta_r = angle between attachment points  
theta_p = angle between rotation points  
theta_s = orientation of the servos  
RD = distance to end effector attachment points  
PD = distance to servo rotation points  
L1 = servo arm length  
L2 = connecting arm length  
z_home = default z height with servo arms horizontal  
servo_min = lower limit for servo arm angle  
servo_max = upper limit for servo arm angle  
servo_mult = multiplier to convert to milliseconds  
p = location of servo rotation points in base frame [x/y][1-6]  
re = location of attachment points in end effector frame [x/y][1-6]  
*/
```

```
const int servo_pin[] = {9, 3, 5, 11, 6, 10}, servo_zero[6] = {1710, 1280, 1700, 1300, 1680, 1300};  
/*
```

```
servo_pin = servo pin assignments,  
servo_zero = zero angles for each servo (horizontal)  
*/
```



```

Servo servo[6];
/*
Servos 0, 2, 4: reversed (+ = down, - = up)
Servos 1, 3, 5: normal (+ = up, - = down)
*/

void setup()
{
  //Serial.begin(9600);
  for(int i = 0; i < 6; i++)
  {
    servo[i].attach(servo_pin[i]);
    servo[i].writeMicroseconds(servo_zero[i]);
  }
  delay(1000);
}

void loop()
{

  static float pe[6] = {0,0,0,radians(0),radians(0),radians(0)}, theta_a[6], servo_pos[6],
    q[3][6], r[3][6], dl[3][6], dl2[6];
  /*
  pe = location and orientation of end effector frame relative to the base frame [sway, surge,
  heave, pitch, roll, yaw)
  theta_a = angle of the servo arm
  servo_pos = value written to each servo
  q = position of lower mounting point of connecting link [x,y,x][1-6]
  r = position of upper mounting point of connecting link
  dl = difference between x,y,z coordinates of q and r
  dl2 = distance between q and r
  */

```

```

for(int i = 0; i < 6; i++)
{
    q[0][i] = L1*cos(-theta_a[i])*cos(theta_s[i]) + p[0][i];
    q[1][i] = L1*cos(-theta_a[i])*sin(theta_s[i]) + p[1][i];
    q[2][i] = -L1*sin(-theta_a[i]);

    r[0][i] = re[0][i]*cos(pe[4])*cos(pe[5]) + re[1][i]*(sin(pe[3])*sin(pe[4])*cos(pe[5]) -
cos(pe[3])*sin(pe[5])) + pe[0];
    r[1][i] = re[0][i]*cos(pe[4])*sin(pe[5]) + re[1][i]*(cos(pe[3])*cos(pe[5]) +
sin(pe[3])*sin(pe[4])*sin(pe[5])) + pe[1];
    r[2][i] = -re[0][i]*sin(pe[4]) + re[1][i]*sin(pe[3])*cos(pe[4]) + z_home + pe[2];

    dl[0][i] = q[0][i] - r[0][i];
    dl[1][i] = q[1][i] - r[1][i];
    dl[2][i] = q[2][i] - r[2][i];

    dl2[i] = sqrt(dl[0][i]*dl[0][i] + dl[1][i]*dl[1][i] + dl[2][i]*dl[2][i]) - L2;

    theta_a[i] += dl2[i];

    theta_a[i] = constrain(theta_a[i], servo_min, servo_max);

    if(i%2 == 1) servo_pos[i] = servo_zero[i] + theta_a[i]*servo_mult;
    else servo_pos[i] = servo_zero[i] - theta_a[i]*servo_mult;
}

for(int i = 0; i < 6; i++)
{
    servo[i].writeMicroseconds(servo_pos[i]);
}
}

```

