Design Competition 2014 Workshop 3: Linkages and Simulation Sensors

Milestone 3 due by Wednesday, 2/26/14, at 5pm, by email or demonstration to Nick

Task:

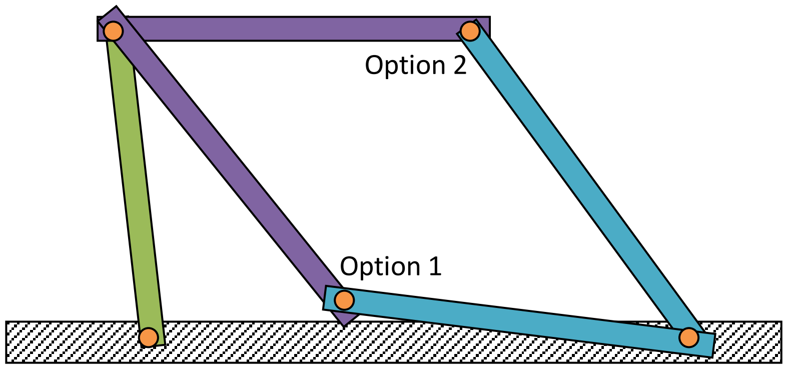
1. Design **a four-bar linkage** using radial ball bearings, mount it to a small RC servo, and use a 555 timer circuit to drive the RC servo.
2. Use the **LIDAR** sensor to create a hall-wandering robot.

Download:

1. DC14Workshop3.zip, containing DClidarControl.m and maze4.

Four-bar linkages

- Most motors create rotational motion. It is common practice to use screws, cams, and mechanisms to convert rotation into other types of elliptical, reciprocating, or linear motion. In this exercise, you will build a set of linkages in a closed-loop to convert the 0 degree to 180 degree motion of an RC servo into something else (your choice!).
- Take a look through the following types of mechanisms:
    - http://kmoddl.library.cornell.edu/model.php?m=234
    - http://mechanicaldesign101.com/category/linkage-animation/
    - http://www.mekanizmalar.com/menu_linkage.html
- A four-bar linkage has one degree of freedom – one input gets transformed into one output. Adding more links add more degrees of freedom, and may require more inputs. Take a look at this interesting whiteboard clock, using a five-bar linkage and two RC servos, and an out of plane servo:
    - http://www.youtube.com/watch?v=iOLFP90DneY
- The danger with all linkages is that there can be two or more possible solutions to some endpoint positions:



- You will want to avoid the positions where a joint can go straight and pop-out into a bad configuration, either by making a linkage that doesn't have more than one endpoint solution, or by including mechanical stops that prevent joints from overextending.

Simulating linkages

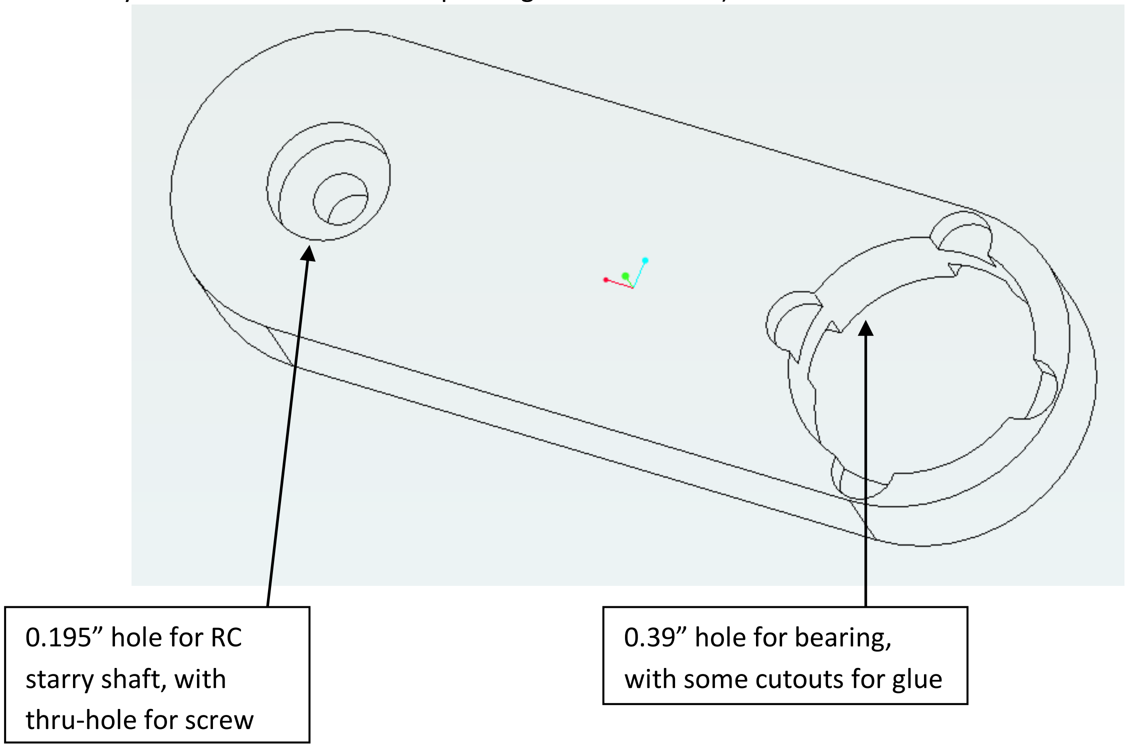
- Most CAD software packages have a simulation mode, where you can define joints as rotational, prismatic, sliding, etc., and then grab and move your parts around to see how the endpoint will move and if any parts will collide with each other.
- To design the actual motion of the device, and choose link lengths, you might want to copy a configuration from a text book, online resource, or existing design. You can quickly prototype the linkage lengths with LEGO (next to the Makerbot in the Mechatronics lab) and play with different lengths. Cardboard and foam core will also do, and you can make the joints with pins or nails.

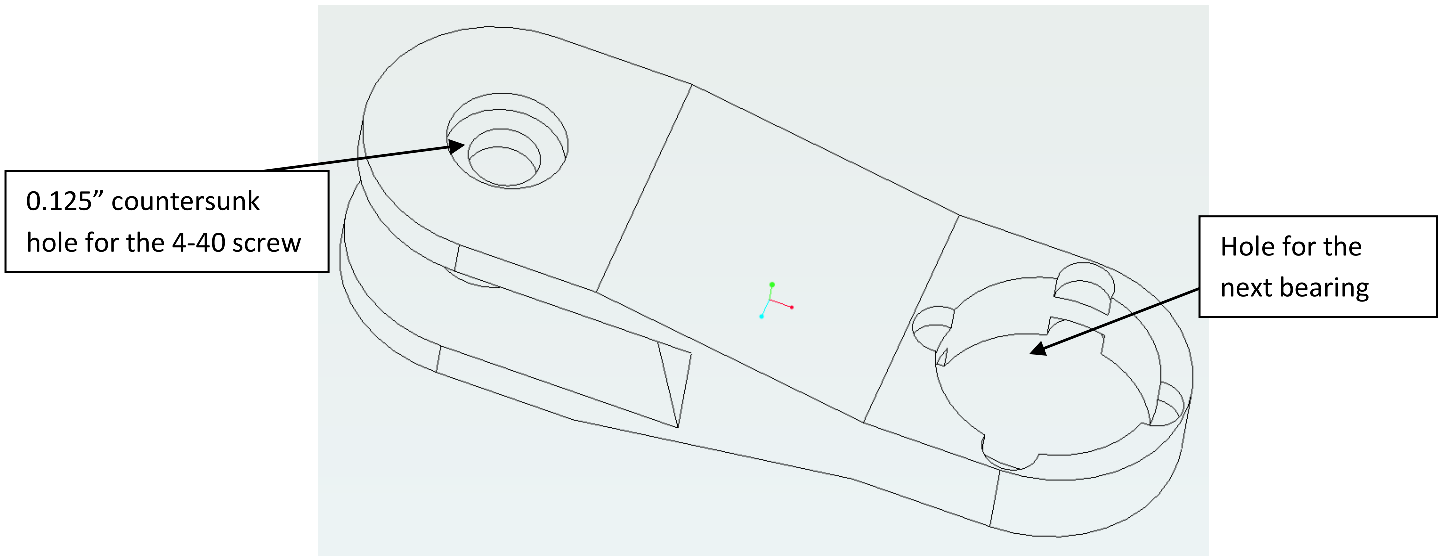
- Simulation with Matlab, Mathematica, and lots of custom and specific programs is possible, but I have not found a single, easy to use, flexible and free program. Try these:
    - http://www.mekanizmalar.com/fourbar01.html
    - Download the Wolfram CDF player (you don't have to register, just hit download) http://demonstrations.wolfram.com/download-cdf-player.html, and run the 4-bar example: http://demonstrations.wolfram.com/PlottingClosedCurvesWithAFourBarLinkage/ then try the Theo Jansen walking linkage, a really cool leg design: http://demonstrations.wolfram.com/ATheoJansenWalkingLinkage/ (also see http://www.youtube.com/watch?v=CufN43By79s and http://www.youtube.com/watch?v=azy-c6QXUCw and http://www.youtube.com/watch?v=CzzAgVOsTKQ)
    - Run Geogebra, a neat little geometry program, http://www.geogebra.org/, and take a look at how to use it to design and simulate linkages at http://www.oit.edu/faculty/randy.shih/mech407/GeoGebra.pdf

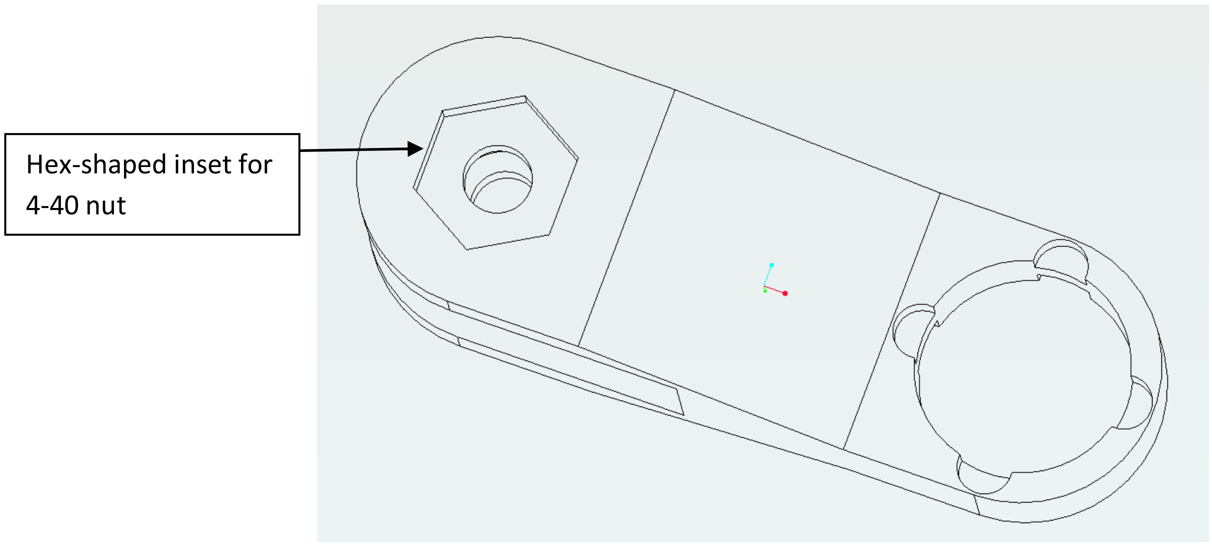
Designing with bearings

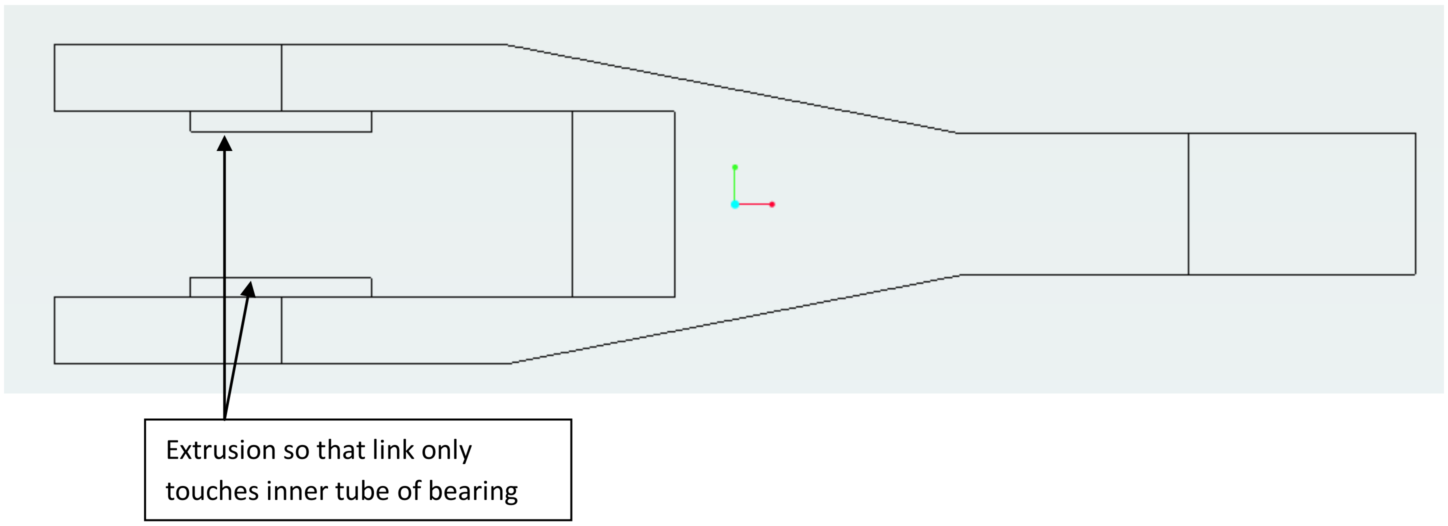
- Linkages are only as good as the connections between the links. The rotations need to be smooth, without friction, sticking, damping, or grinding, and they need to be tight, no wiggling, play, or loosening.
- The key to a good link connection is a bearing – in our case, a radial ball bearing. This type of bearing consists of an inner steel tube and an outer steel tube, with spherical ball bearings and grease in between, and some washers to keep the balls from falling out.

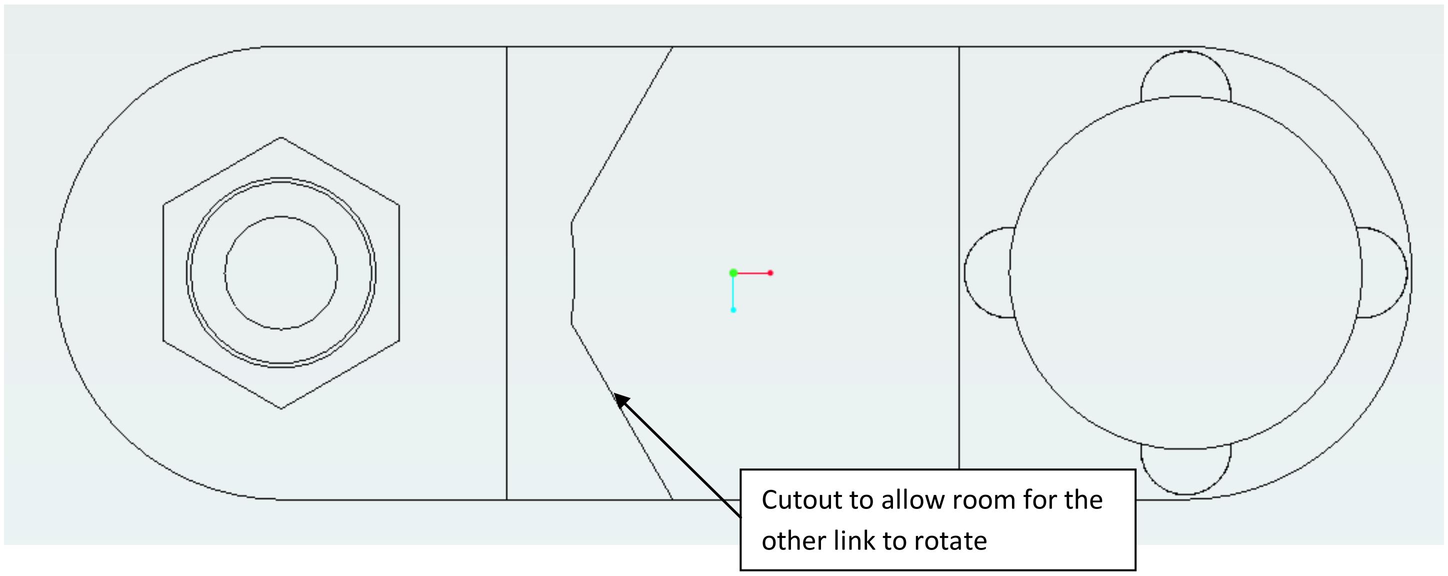
- Our bearings are 1/8" inner bore diameter and 3/8" outer diameter, and 5/32" thick. A 4-40 screw is almost 1/8" in diameter and can be used as the rotation shaft. You can embed a 4-40 nut into your link to hold it in place.
- A good bearing connection will imbed the bearing in one link, and connect the other link to the inner tube, without touching the washers or other link. Here is one possible design (all holes are 0.015" larger than necessary to account for Makerbot printing them too small):


0.195" hole for RC starry shaft, with thru-hole for screw
0.39" hole for bearing, with some cutouts for glue

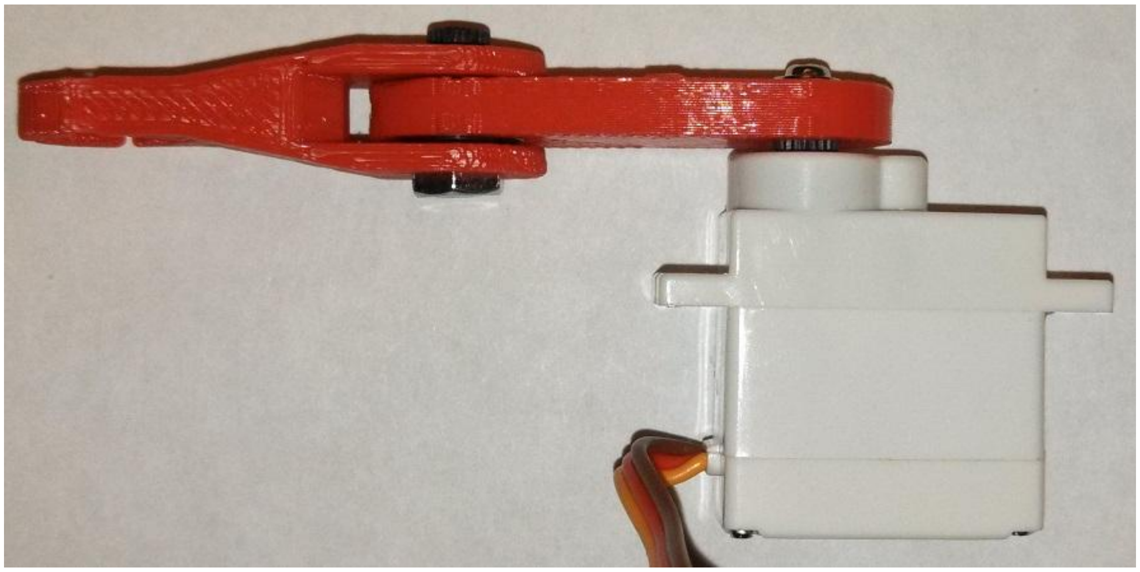

0.125” countersunk hole for the 4-40 screw
Hole for the next bearing


Hex-shaped inset for 4-40 nut


Extrusion so that link only touches inner tube of bearing

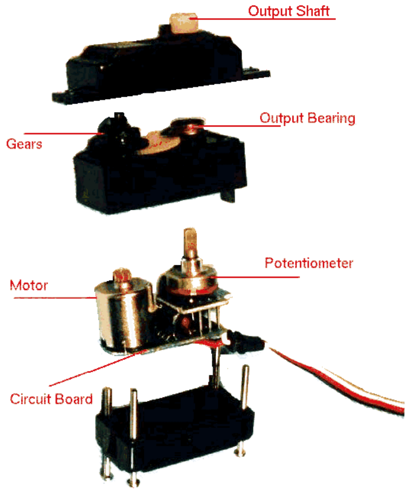


What it looks like in real life:

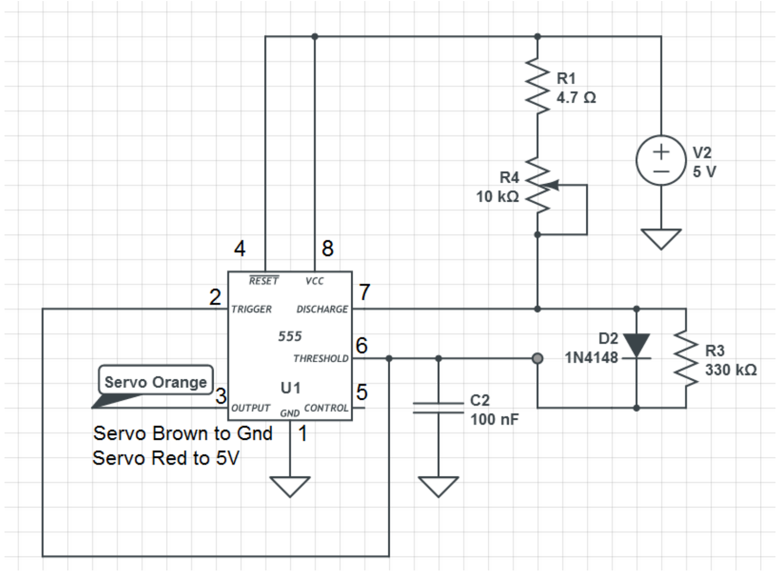


RC servo

- A servo typically refers to a motor with position sensor and electronic controller all wrapped up in one package. An RC servo is an inexpensive motion control system designed for remote control cars, planes, and boats. They come in many sizes, torques, speeds, and prices.
- Typical construction uses a small brushed DC motor, many gears, and a potentiometer for position sensing:



- A small controller board takes a strange pulse wave and converts it to a desired position, and steers the motor to that position. Because potentiometers have limited turning range, most servos can only turn 180 degrees. A mechanical stop prevents them from moving outside of this range, and commanding one to move too far will burn out the motor, because an RC servo has no way to tell you it can't get to where you are asking it to go.
  - One cool thing you can do here is open up the servo, cut out the mechanical stop, remove the potentiometer, and solder some wires in its place. If you wire in a voltage divider, you make a speed controller, and can have the servo rotate indefinitely (called a 360 degree RC servo). Or you can build a linkage, embed a potentiometer somewhere, and the servo will work as before, but with a new position reference.
- The pulse wave comes from the way radio remote controls send signals, which is dated and a little strange. The pulse occurs every 20ms (50Hz), with a pulse width of 0.5ms representing 0 degrees, and a pulse width of 2.5ms representing 180 degrees (either poor quality control or different manufacturers using slightly different signals means you often have to verify these numbers). There is a nice 555 timer circuit that will read a potentiometer and convert its angle to a signal to drive an RC servo to the same angle:



Roughly based on http://www.555-timer-circuits.com/servo-tester.html

- The output shaft of an RC servo is a starry-shaped connector that you can try to connect to directly, or it will mate to a “servo horn” and you can mount to the horn. Every RC servo comes with a little bag of horns, and some screws that can be used to securely mount the horn to the servo:

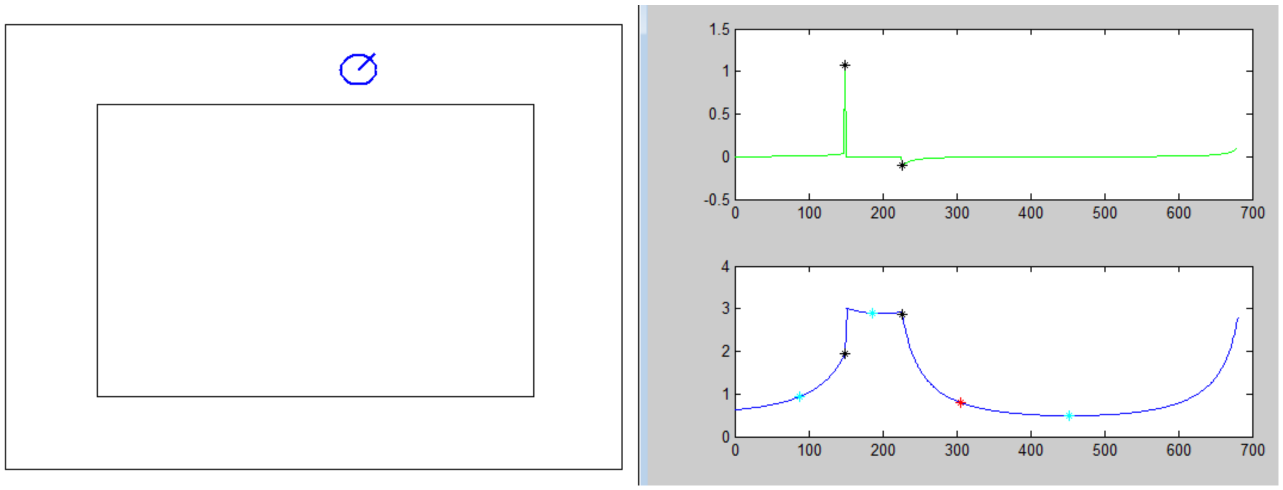
Four-bar linkage assignment

- Design a 4-bar linkage configuration and link lengths in a simulator. CAD the links and use bearings at three of the joints, and drive the fourth joint with the RC servo. Build the 555 timer circuit and control the mechanism with a potentiometer. (Note RC servos cost $3 and each bearing costs $1 – cheap enough to mess around with without worrying about the price, but please try to design so that you can reuse these parts and not be wasteful).

Now, more robot simulation…

LIDAR

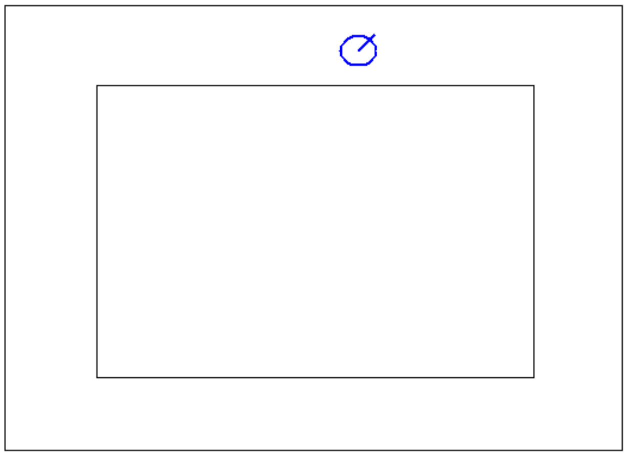
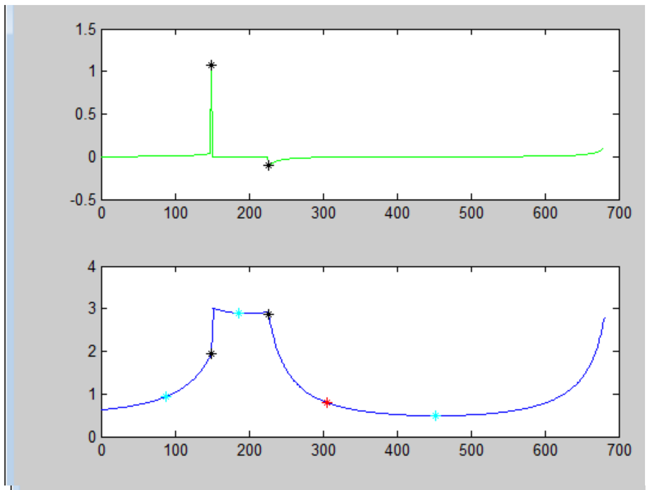
- The LIDAR sensor is like the sonar sensor, but gives distance readings at from -120 degrees to +120 degrees around the robot, in 681 measurements. The maximum range of the LIDAR is 4, and the minimum is 0.02 (note: you can change these parameters in CreateRobot.m), so a typical LIDAR reading is an array that looks like the blue trace:

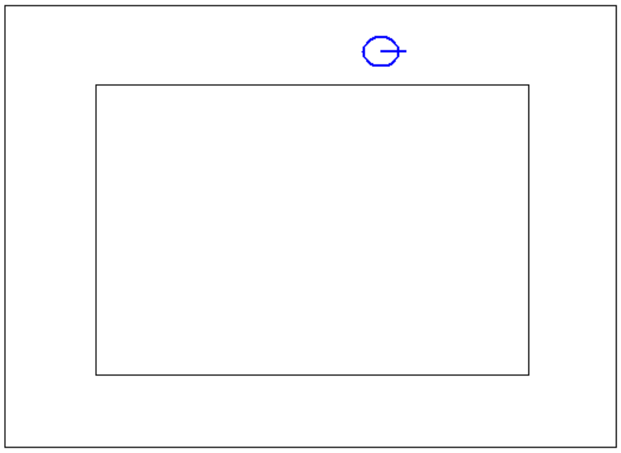
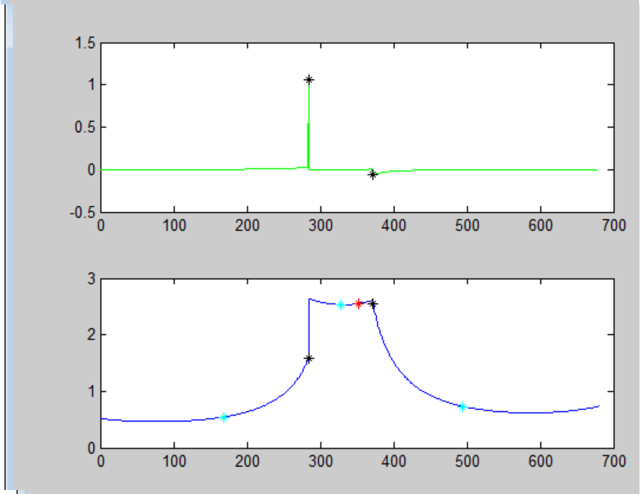


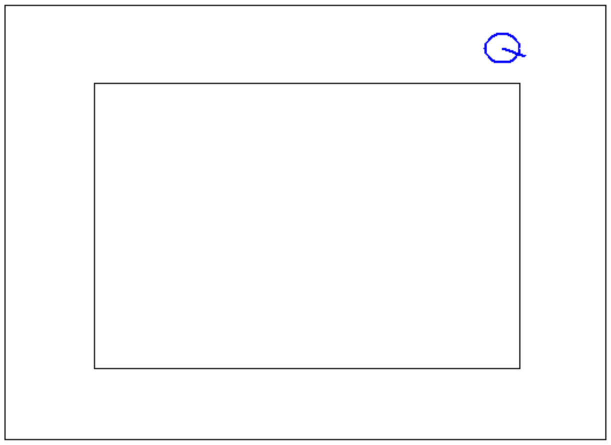
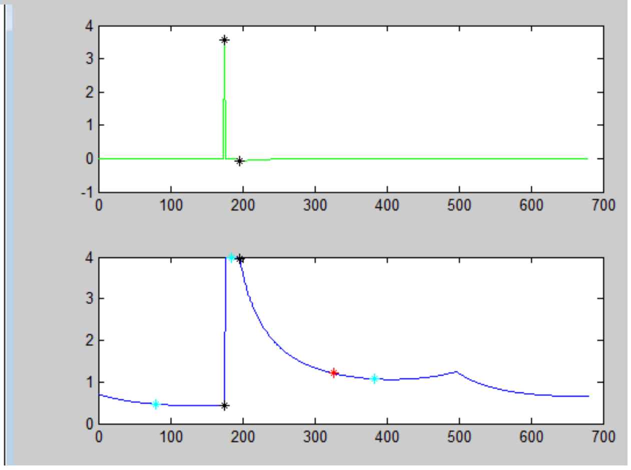
- Using the LIDAR is harder than the other sensors because there is so much data – you will need to run a for loop to analyze it, and when you are writing and testing your algorithms you will probably need to plot the data and the output of your analysis every time you read the LIDAR to know what is going on. It is helpful to place the robot in a specific location and run your controller for only one loop cycle, rather than have it update continuously and overwrite your plots. See DClidarControl.m

Using LIDAR for hall-following

- The goal of hall-following is to stay in the middle of the hallway as you drive around in a loop. You might be able to do this with sonar as well, but let’s see what all this data can provide.
- A typical LIDAR read will provide the following (in Blue):

-  

-  

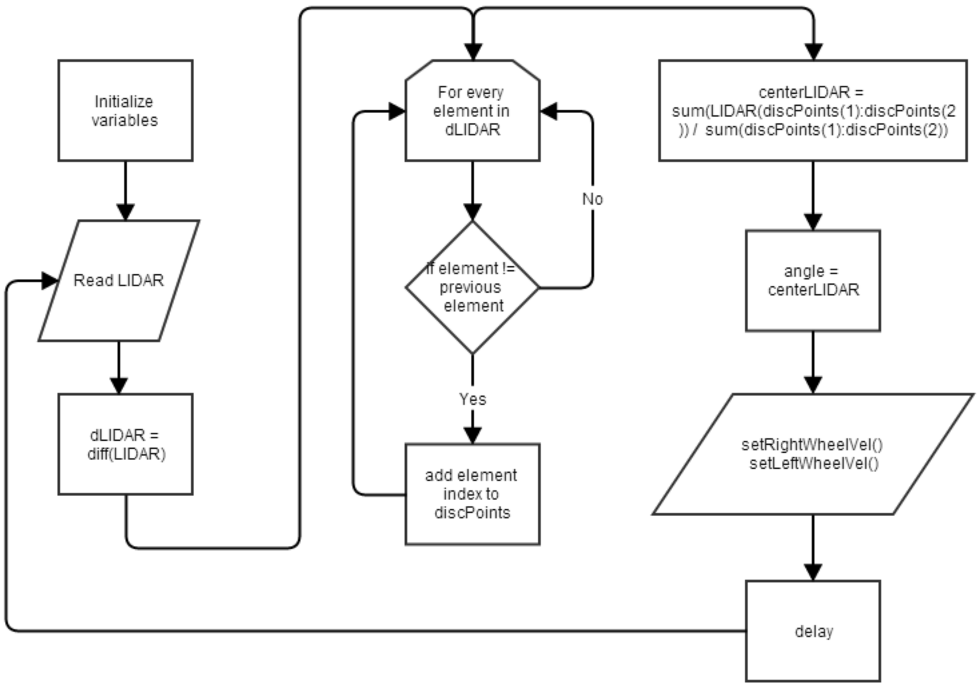
-  

- How do we decide what to do with the wheel speeds? I can think of a bunch of methods to try out, some simple, some complicated. It's hard to know how complicated you need to go: if the hallway is tight and straight, there are not many different categories of LIDAR graphs to have to deal with. But if there are ways for the robot to get stuck in a corner, or see a door, you will have to add more cases in your controller to try to identify those problems and deal with them. Try out the following method, but keep in mind there are easier ways to do it that might work better, and more complicated methods that can be applied to different mazes.
- Method 1: Calculate the center of mass of the LIDAR array, and drive towards it (represented by the Red stars in the graphs above)
- Method 2: Take a derivative of the data with respect to angle (the Matlab diff() function, in Green above) and calculate the center of mass of the LIDAR array between the min and max of the LIDAR diff() (min and max in Black stars above)

- Method 3: Calculate the center of mass between the start of the LIDAR array to the first Black star, between the Black stars, and from the second Black star to the end of the array (shown as Cyan stars) and drive towards the largest of the center of masses.
- Now all you need is to turn the center of mass into wheel velocities. Keep a constant forward velocity, and scale the left and right wheel speed by the angle you want to drive towards.

Visualizing the algorithm

- This algorithm is complicated enough to need to draw a flow chart to visualize, something like:


Initialize variables
Read LIDAR
dLIDAR = diff(LIDAR)
For every element in dLIDAR
If element != previous element
No
Yes
add element index to discPoints
centerLIDAR = sum(LIDAR(discPoints(1):discPoints(2)) / sum(discPoints(1):discPoints(2))
angle = centerLIDAR
setRightWheelVel()
setLeftWheelVel()
delay

Simulation Assignment

- Create a hall-following controller to navigate maze4.
- Extra credit: scale the forward velocity by how sure you are in the direction you should drive – for example, if the LIDAR data is “flat”, go slower so there is less distance covered before the next loop.

In Workshop 4 we will learn about different types of robots legs and gaits, and you will make a maze solving robot for a maze with doors.