# Mobile Robotics Exam #3

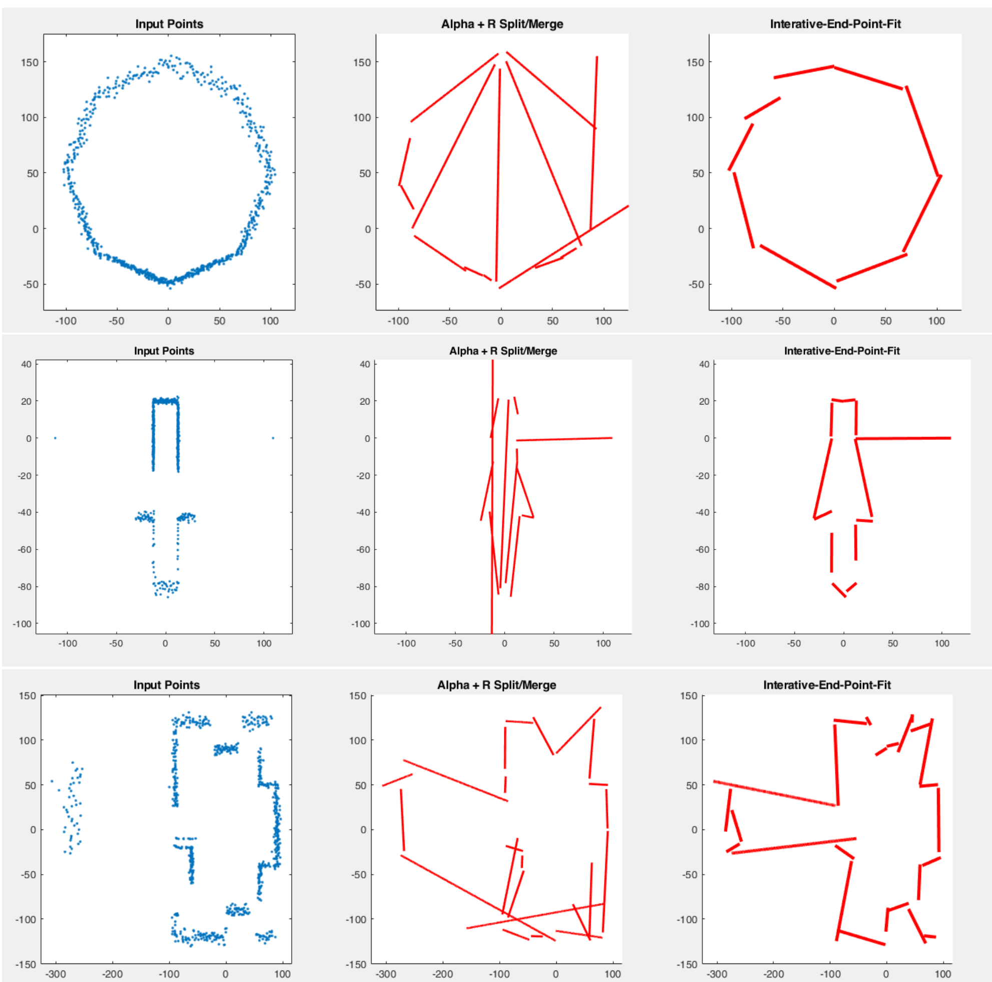
Charlie Coleman

## Wall Detection

For the wall detection code, I ended up writing 2 separate algorithms. Firstly, I attempted the split-and-merge algorithm using the equations for $\alpha$ & r to generate our line segments. I used Euclidean distance from all points to the line to find the error, then split the line segment at the point furthest from the line. This method did not accomplish a very good result, so I implemented a second method.

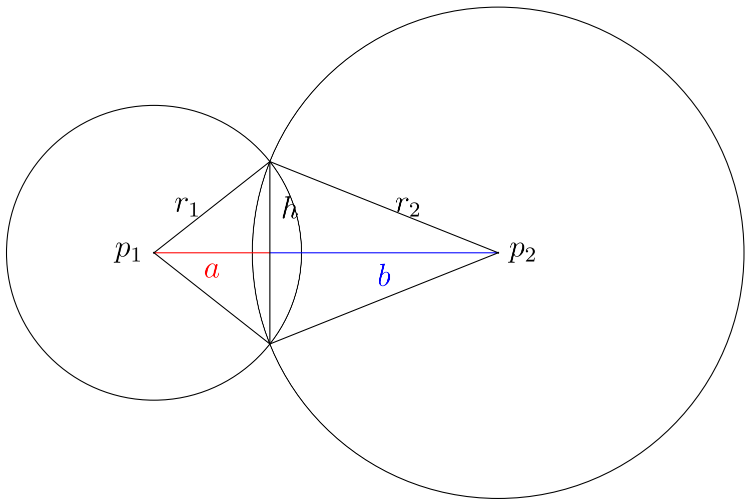
For the second attempt, I used Iterative-End-Point-Fit. This is very simliar to split-and-merge, but in this algorithm, the line sample line was drawn directly from the first to the last point. Then the error is calculated and the methodology from above is followed.

Both functions are written using recursion to split the lines until the general shape of the field is recovered. Outputs from the 3 test point sets are shown below.



## Trilateration

To find our position, first I needed to find the intersections of every combination of two circles. To do this, I attempted to use the methodology discussed in class, where we use the equations of the circles to find their common points. I ran into some issues given the amount of square roots within the calculations, where in each I had to handle the negative and positive result. This was causing me some issues so I decided to go with a more trigonometric approach to the problem. Below is a diagram showing the line segments that I used to calculate the points of intersection for each circle combination.




Firstly, I had to check that the circles are close enough to overlap. This is an easy calculation, as you simply check if $d > r_1 + r_2$. If it turns out that the circles are too far apart, I used the point between the circles (proportional to the radius of each circle) as the only 'intersection point'.

If the circles do overlap, we can find the point between the two intersection points. Using the Pythagorean Theorem, we can write the equations:

$$a^2 + h^2 = r_1^2 \text{ and } b^2 + h^2 = r_2^2$$

From these we can solve for a $(d = a + b)$, and get

$$a = \frac{r_1^2 - r_1^2 + d^2}{2d}$$

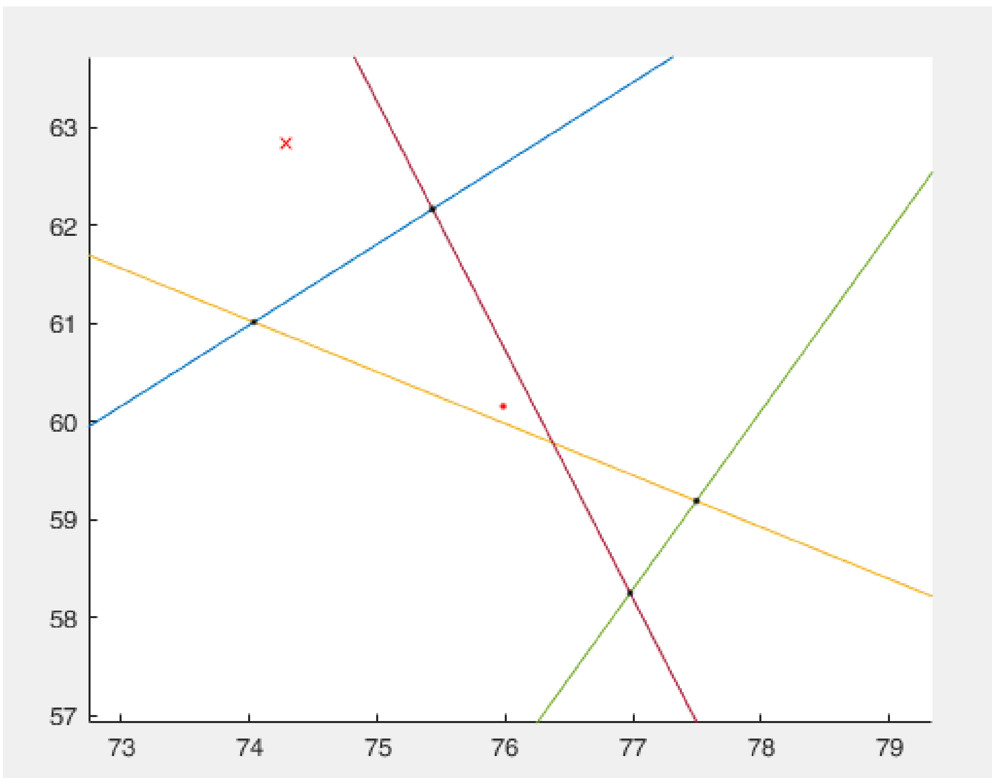
With $a$, we can solve the right triangle with sides $a$, $h$, and $r_1$ to get the intersection points.

With these two intersection points, I calculate their distance to the third circle. I compare these distances to the distance from the third beacon to the goal. The intersection point with a distance closer to distance of the third beacon. After looping through all three combinations of circles, I'm left with 3 'good' intersection points. I use the centroid of these 3 points as my return value.

## Triangulation

To estimate our position in problem three, I found the intersection point of each line with each of the other lines. I placed my beacons in the corners of the field. This caused some errors when I used the intersection point with beacons opposite each other on the grid. Because of this, I only used the two intersection points of the next beacon around the board. Using this method, I obtained four 'good' intersection points and my estimate was the centroid of these points. Below is an example output of this method. The black points are the good intersection points, the red dot is the estimated position and the red x is the actual position.

I did not use the inscribed angle method used in class because the angle of the robot was not needed in our result, only the position.



# Code

Problem 1: exam3_q1.m

```
close all
clear all

load estimate_set3.mat
%load estimate_set2.mat
%load estimate_set3.mat

theta = dist(:,1);
rho = dist(:,2);

var = (rho.^2)./(10^3);
w = 1./var;

points = [rho.*cos(theta), rho.*sin(theta)];
bounds = [min(points(:, 1)) - 20, max(points(:, 1)) + 20, min(points(:, 2)) - 20, max(points(:, 2)) + 20];

p1 = subplot(1,3,1);
plot(rho.*cos(theta),rho.*sin(theta), '.');
hold on;
title(p1, 'Input Points');
axis(bounds);

p2 = subplot(1,3,2);
title(p2, 'Alpha + R Split/Merge');
hold on

mid = floor( length(points) / 2 );
e = length(points);

maxErr = 16;

[ms1, bs1] = arSAM(w(1:mid), rho(1:mid), theta(1:mid), points(1:mid, :), maxErr);
[ms2, bs2] = arSAM(w(mid:end), rho(mid:end), theta(mid:end), points(mid:end, :), maxErr);
axis(bounds)

p3 = subplot(1,3,3);
title(p3, 'Interative-End-Point-Fit');
hold on

[ms3, bs3] = SAM(w, rho, theta, points, maxErr);

axis(bounds);

function [ms, bs] = arSAM(w, rho, theta, points, maxDist)
    [alpha, r] = calcAlphaR(w, rho, theta);
    [m, b] = calcSlopeIntercept(alpha, r);
    ds = distToLine(points, m, b);
    foundBreak = 0;
    [val, index] = max(ds);
    while not(foundBreak) && index <= length(ds)
        if (index >= length(ds)) || (val < maxDist) || (ds(index+1) > maxDist)
            foundBreak = 1;
        else
            ds(index) = 0;
        end
        [val, index] = max(ds);
    end

    if index <= 10 || length(w(index:end)) <= 10
        middle = ceil(length(w)/2);
        w = circshift(w, middle - index);
        rho = circshift(rho, middle - index);
```

```
        theta = circshift(theta, middle - index);
        points = circshift(points, middle - index);
        index = middle;
    end

    minPoints = 10;
    if val > maxDist && length(w(1:index)) > minPoints && length(w(index:end)) > minPoints
        [m1, b1] = arSAM(    w(1:index),      rho(1:index),      theta(1:index),      points(1:index, :), maxDist);
        [m2, b2] = arSAM(w(index+1:end), rho(index+1:end), theta(index+1:end), points(index+1:end, :), maxDist);

        ms = horzcat(m1, m2);
        bs = horzcat(b1, b2);
    else
        ms = m;
        bs = b;
        x = -400:0.1:400;
        y = m * x + b;
        xmax = max(points(:, 1));
        xmin = min(points(:, 1));
        ymax = max(points(:, 2));
        ymin = min(points(:, 2));
        if xmax-xmin > ymax-ymin
            plot(x(x < xmax & x > xmin), y(x < xmax & x > xmin), 'r', 'LineWidth', 2);
        else
            plot(x(y < ymax & y > ymin), y(y < ymax & y > ymin), 'r', 'LineWidth', 2);
        end
    end
end

function [ms, bs] = SAM(w, rho, theta, points, maxDist)
    m = ( points(1, 2) - points(end, 2) ) / ( points(1, 1) - points(end, 1) );
    if isnan(m)
        m = 10000;
    end
    b = points(1, 2) - m * points(1, 1);
    ds = distToLine(points, m, b);
    [val, index] = max(ds);

    if val > maxDist && length(w) > 25
        [m1, b1] = SAM(    w(1:index),      rho(1:index),      theta(1:index),      points(1:index, :), maxDist);
        [m2, b2] = SAM(w(index+1:end), rho(index+1:end), theta(index+1:end), points(index+1:end, :), maxDist);
        ms = horzcat(m1, m2);
        bs = horzcat(b1, b2);
    else
        ms = m;
        bs = b;
        x = -400:0.1:400;
        y = m * x + b;
        xmax = max(points(:, 1));
        xmin = min(points(:, 1));
        ymax = max(points(:, 2));
        ymin = min(points(:, 2));
        if xmax-xmin > ymax-ymin
            plot(x(x < xmax & x > xmin), y(x < xmax & x > xmin), 'r', 'LineWidth', 3);
        else
            plot(x(y < ymax & y > ymin), y(y < ymax & y > ymin), 'r', 'LineWidth', 3);
        end
    end
end

function [alpha, r] = calcAlphaR(w, rho, theta)
    sum2 = 0;
    sum4 = 0;
```

```
    sum1 = sum(w .* (rho.^2) .* sin(2*theta));
    sum3 = sum(w .* (rho.^2) .* cos(2*theta));
    sumW = sum(w);
    for i = 1:length(w)
        for j = 1:length(w)
            sum2 = sum2 + ( w(i) * w(j) * rho(i) * rho(j) * cos( theta(i) ) * sin( theta(j)
                ) );
            sum4 = sum4 + ( w(i) * w(j) * rho(i) * rho(j) * cos( theta(i) + theta(j) ) );
        end
    end

    % sum1 = SUM w_i p_i^2 sin(2 theta_i)
    % sum2 = SUM SUM w_i w_j p_i p_j cos(theta_i) sin(theta_j)
    % sum3 = SUM w_i p_i^2 cos(2 theta_i)
    % sum4 = SUM SUM w_i w_j p_i p_j cos(theta_i + theta_j)
    % alpha = 1/2 * atan( (sum1 - 2/(sum w) * sum2) / (sum3 - 1/(sum w) * sum4)

    alpha = 1/2 * atan2( (sum1 - (2/sumW * sum2)), (sum3 - (1/sumW * sum4))) + pi/2;

    % sum5 = SUM w_i p_i cos( theta_i - alpha)

    sum5 = sum(w .* rho .* cos(theta - alpha));
    r = sum5 / sumW;
end

function [m, b] = calcSlopeIntercept(alpha, r)
    p1 = [r * cos(alpha), r * sin(alpha)];
    p2 = p1 + [10 * cos(alpha + pi/2), 10 * sin(alpha + pi/2)];

    m = (p2(2) - p1(2))/(p2(1) - p1(1));
    b = p2(2) - (m * p2(1));
end

function d = distToLine(p, m, b)
    % d = max( abs((m*p(:, 1)+b) - p(:, 2)), abs((p(:, 2) - b)/m - p(:, 1)));
    d = abs( -m * p(:, 1) + p(:, 2) - b) / sqrt(m^2 + 1);
end

function d = distBetweenPoints(p1, p2)
    d = sqrt( (p1(2) - p2(2))^2 + (p1(1) - p2(2))^2 );
end

function md = mahalanobis(a, r, theta, p)
    md = (a - theta(:)).^2 + (r - p(:)).^2;
end
```

## Problem 2: exam3_q2.m

```
function [x, y] = exam3_q2()
    points = [[0,0];[50,100];[100,0]];
    dists = zeros(3, 1);
    colors = ['r', 'g', 'b'];
    intersections = zeros(3,2);
    ref = [[0,3,2];[3,0,1];[2,1,0]];

    for i = 1:3
        p = points(i, :);
        dists(i) = get_dist(p(1), p(2));
    end

    figure
    axis([-50 200 -50 200]);
    hold on

    %draw circles
    theta = 0:pi/50:2*pi;
    px = 1*sin(theta);
    py = 1*cos(theta);

    for i = 1:3
        p = points(i, :);
        d = dists(i);
        px1 = px * d + p(1);
        py1 = py * d + p(2);

        px1(end+1) = px1(1);
        py1(end+1) = py1(1);

        plot(px1, py1, colors(i));
        plot(p(1), p(2), horzcat(colors(i), '.'));
    end

    for i = 1:2
        for j = i+1:3
            p0 = points(i, :);
            p1 = points(j, :);
            r0 = dists(i);
            r1 = dists(j);
            refN = ref(i, j);
            refP = points(refN, :);
            refD = dists(refN);
            d = sqrt((p0(1) - p1(1))^2 + (p0(2) - p1(2))^2);

            a = (r0^2 - r1^2 + d^2)/(2*d);
            p2 = zeros(2,1);
            p2(1) = p0(1) + a * (p1(1) - p0(1))/d;
            p2(2) = p0(2) + a * (p1(2) - p0(2))/d;

            if d > (r0 + r1)
                x1 = p2(1);
                y1 = p2(2);

                x2 = p2(1);
                y2 = p2(2);
            else
                h = sqrt(r0^2 - a^2);

                x1 = p2(1) + h * (p1(2) - p0(2))/d;
                y1 = p2(2) - h * (p1(1) - p0(1))/d;

                x2 = p2(1) - h * (p1(2) - p0(2))/d;
                y2 = p2(2) + h * (p1(1) - p0(1))/d;

            end
```

```
            dist1 = sqrt((x1 - refP(1))^2 + (y1 - refP(2))^2);
            err1 = abs(refD - dist1)/refD;

            dist2 = sqrt((x2 - refP(1))^2 + (y2 - refP(2))^2);
            err2 = abs(refD - dist2)/refD;

            if err1 < err2
                intersections(i + (j - 2), :) = [x1, y1];
            else
                intersections(i + (j - 2), :) = [x2, y2];
            end
        end
    end

    avgP = [mean(intersections(:, 1)), mean(intersections(:, 2))];
    plot(avgP(1), avgP(2), 'k*');

    x = avgP(1);
    y = avgP(2);
```

## Problem 3: exam3_q3.m

```
function [x, y] = exam3_q3()

    points = [[0,0]; [0, 100]; [100, 100]; [100, 0]];
    angles = zeros(4, 1);
    mvals = zeros(4, 1);
    bvals = zeros(4, 1);

    intersections = zeros(4, 2);

    figure
    axis([-25 125 -25 125]);
    hold on

    for i = 1:length(points)
        p0 = points(i, :);
        angles(i) = get_angle(p0(1), p0(2), 0);
        a = angles(i);
        p1 = p0 + [10*cos(a), 10*sin(a)];

        m = (p1(2) - p0(2))/(p1(1) - p0(1));
        b = p1(2)-m*p1(1);
        xs = 0:0.1:100;
        ys = m*xs+b;

        mvals(i) = m;
        bvals(i) = b;

        plot(xs(ys < 125 & ys > -25), ys(ys < 125 & ys > -25));
        plot(p0(1), p0(2), 'x');
    end

    for i = 1:length(points)
        p0 = points(i, :);
        a = angles(i);
        refN = mod(i + 1, 4);
        if refN == 0
            refN = 4;
        end

        m1 = mvals(i);
        m2 = mvals(refN);
        b1 = bvals(i);
        b2 = bvals(refN);

        x1 = -(b1 - b2)/(m1 - m2);
        y1 = m1 * x1 + b1;

        plot(x1, y1, 'k.');
        intersections(i, :) = [x1, y1];
    end

    x = mean(intersections(:, 1));
    y = mean(intersections(:, 2));

    plot(x, y, 'r.');
```