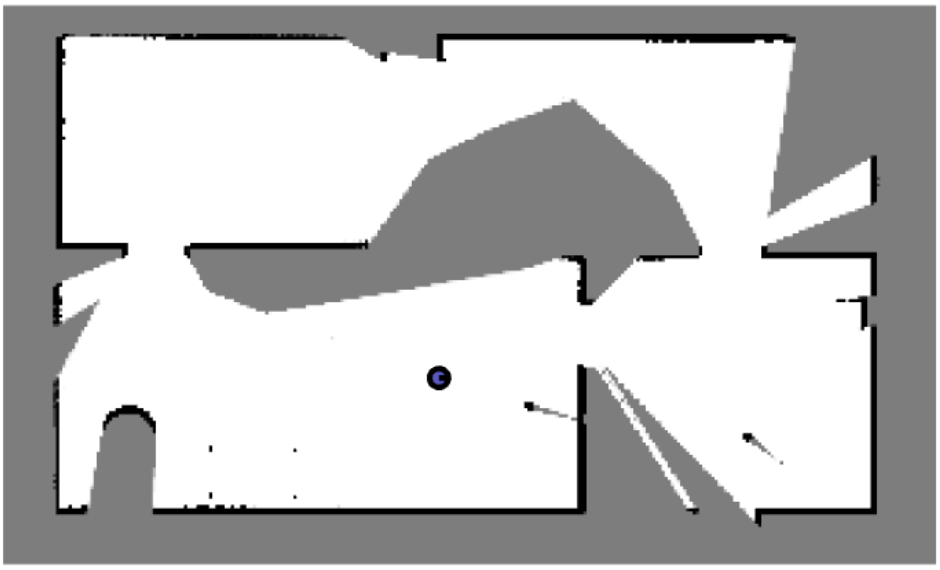
Assignment 2 Report
ROB521: Mobile Robotics and Perception

Name: Malcolm MacKay **Student number: 1002423308**

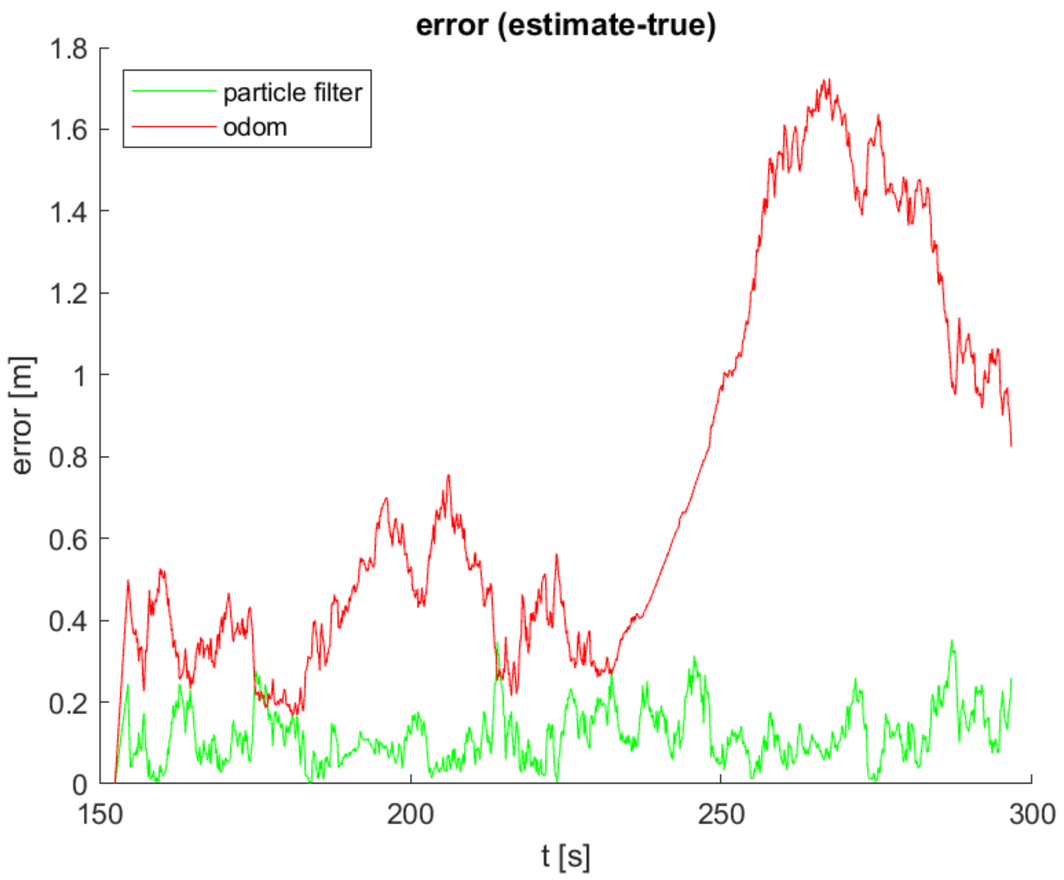
Question 1:

My results for question 1 look identical to the solution. In my first few iterations, I found that I was getting the right shape, but it was marking everything in black (obstacle) as opposed to grey for unexplored territory and black for obstacles. We can see that spaces past an obstacle that the robot couldn't see are marked as grey, for example in the bottom left of the image. On anomaly I found is the single line of white in the bottom right corner- since we always have a cone of values, it's peculiar to see a single line. My prediction is that there was a limited amount of time where the camera was facing that direction, so it quickly identified the obstacle, but would take more time to fill in the blank space.



Question 2:

My results for question 2 look identical to the solution. The results make sense, we can see that the odometric estimation error is significantly higher and grows exponentially (as expected), while the particle filter error stays relatively constant as time goes on. In the video, we can see that the point cloud gets tighter around the robot as time goes on, since the robot is seeing landmarks it recognizes and re-estimating the place of the particles. At the end of the video, when the robot isn't moving for a period of time, the point cloud begins to disperse and get less tight, likely because the robot isn't receiving more information to update it's estimates, leading to a larger variance.



Appendix (Code)

Question 1:

```
% ------insert your occupancy grid mapping algorithm here------
%patch 1 from A1
if abs(omega_interp(i)) >= 0.1
    continue
end

% Constants
alpha = 1;
beta = -1;

%create rotation matrix and vector that go from the initial frame to the
%robot frame
C_B = [cos(theta_interp(i)) -sin(theta_interp(i));
       sin(theta_interp(i)) cos(theta_interp(i))];
r_B = [(x_interp(i) - 0.1 * cos(theta_interp(i))); %patch 2 from a1
       (y_interp(i) - 0.1 * sin(theta_interp(i)))];


for j=1:npoints
    %check if laser reading in range
    if y_laser(i,j) < r_min_laser || y_laser(i,j) > r_max_laser
        continue
    end

    %slice vector into grid sized boxes
    slice = (r_min_laser:ogres:y_laser(i,j));

    for q=1:size(slice, 2)
        %calculate vector from robot to slice of laser
        r_L = [slice(q)*cos(angles(j));
               slice(q)*sin(angles(j))];

        %transform back to world frame
        r_W = C_B * r_L + r_B;

        %place on the grid
        r_G = [ceil((r_W(1)-ogxmin)/ogres);
               ceil((r_W(2)-ogymin)/ogres)];
        if isnan(r_G(1)) || isnan(r_G(2))
            continue
        end

        %check if on grid (End of line says (r_G(2) < 1))
        if (r_G(1) > ognx) || (r_G(1) < 1) || (r_G(2) > ogny) || (r_G(2) < 1)
           continue
        end

        if q < size(slice, 2) - 1
            %if before obstacle, add beta and move on to next laser
            oglo(r_G(2), r_G(1)) = oglo(r_G(2), r_G(1)) + beta;
        else
            %if at obstacle, add alpha and continue
            oglo(r_G(2), r_G(1)) = oglo(r_G(2), r_G(1)) + alpha;
        end
    end
end

%threshholding
```

```
ogp_next = exp(oglo)./(1 + exp(oglo));%new ogp after scan

for y=1:size(ogp,1)
    for x=1:size(ogp,2)
        if (ogp_next(y,x) >= 0.5) && (ogp_next(y,x) >= ogp(y,x))
            ogp(y,x) = ogp_next(y,x); %update if more likely than prior
        end
        if (ogp_next(y,x) < 0.5) && (ogp_next(y,x) < ogp(y,x))
            ogp(y,x) = ogp_next(y,x);
        end
    end
end
% ------end of your occupancy grid mapping algorithm-------
```

Question 2:

```
% ------insert your particle filter weight calculation here ------

%create rotation matrix and vector that go from the initial frame to the
%robot frame
C_B = [cos(theta_interp(i)) -sin(theta_interp(i));
       sin(theta_interp(i)) cos(theta_interp(i))];
r_B = [(x_interp(i) - 0.1 * cos(theta_interp(i))); %patch 2 from a1
       (y_interp(i) - 0.1 * sin(theta_interp(i)))];

if y_laser(i,j) < r_min_laser || y_laser(i,j) > r_max_laser
    continue
end

slice = (r_min_laser:ogres:y_laser(i,j));

for q=1:size(slice, 2)

    %calculate vector to slice of laser (robot frame)
    r_L = [slice(q)*cos(angles(j));
           slice(q)*sin(angles(j))];

    %transform back to world frame
    r_W = C_B * r_L + r_B;

    %place on the grid
    r_G = [ceil((r_W(1)-ogxmin)/ogres);
           ceil((r_W(2)-ogymin)/ogres)];

    %do the same calculation but on the particle
    %calculate vector to particle in world frame
    p_W = T * [r_L; 1];

    %place on the grid
    p_G = [ceil((p_W(1)-ogxmin)/ogres);
           ceil((p_W(2)-ogymin)/ogres)];

    %extra debugging
    if isnan(p_G(1)) || isnan(p_G(2)) || isnan(r_G(1)) || isnan(r_G(2))
        continue
    end
    %double check if points are  on grid
    if (r_G(1) > ognx) || (r_G(1) < 1) || (r_G(2) > ogny) || ...
       (r_G(2) < 1) || (p_G(1) > ognx) || (p_G(1) < 1) || (p_G(2) > ogny) || (p_G(2) < 1)
       continue
    end

    %check if there is an obstacle observed there (based on
    %probability)
    if (ogp(p_G(2), p_G(1)) < 0.5) && (ogp(r_G(2), r_G(1)) < 0.5)
        % Update particle weight using guassian distribution
%the line below-> exp(-(((r_G(1) - p_G(1))^2 + (r_G(2) - p_G(2))^2) / (2 * 100^2)));
        w_particle(n) = w_particle(n) * exp(-(((r_G(1) - p_G(1))^2 + (r_G(2) - p_G(2))^2) / (
    end

end
% ------end of your particle filter weight calculation-------
```