UNIVERSITY OF AMSTERDAM

Solutions for Assignment 3

Luca Simonetto - 11413522
Probabilistic Robotics

September 25, 2017

Question 1

a)

In order to determine the new pose of the robot, given the input pose and the odometry measurements, equation (5.40) from the book has been used.

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \hat{\delta}_{trans} cos(\theta + \hat{\delta}_{rot1}) \\ \hat{\delta}_{trans} sin(\theta + \hat{\delta}_{rot1}) \\ \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \end{bmatrix}$$

As the measurements have to be assumed to be exact, there are no error terms. The exact pose after the first measurement is:

$$\begin{bmatrix} x^1 \\ y^1 \\ \theta^1 \end{bmatrix} = \begin{bmatrix} 2.954 \\ 0.521 \\ 20.000 \end{bmatrix}$$

Combining it with the second measurement, we have the end pose:

$$\begin{bmatrix} x^2 \\ y^2 \\ \theta^2 \end{bmatrix} = \begin{bmatrix} 12.954 \\ 0.521 \\ -10.000 \end{bmatrix}$$

b)

This question requires to plot the expected robot poses after the first measurement, knowing that the values of $\hat{\delta}$ are subject to errors.

As it can be seen from Figure 1, the position estimates of the robot (starting from [0, 0]) are four, each one having two different orientations: this emerges from the fact that an initial and final rotation could be swapped and still resulting in the same pose.

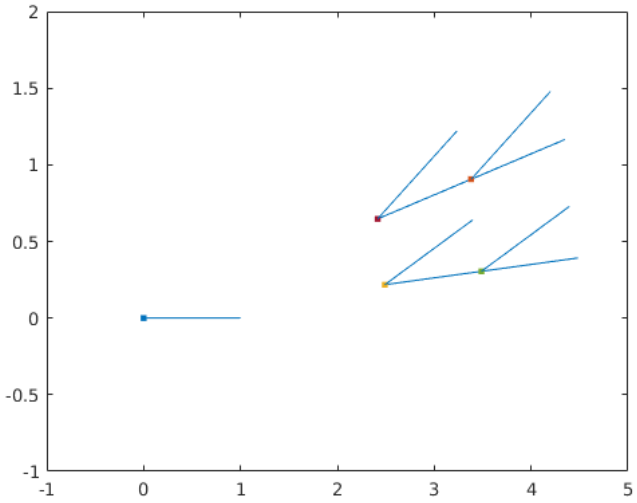

Figure 1: Pose estimates for the first noisy measurement.

c)

This question requires to plot the poses of the same robot but after applying the second measurement to the eight different poses determined in the past question.

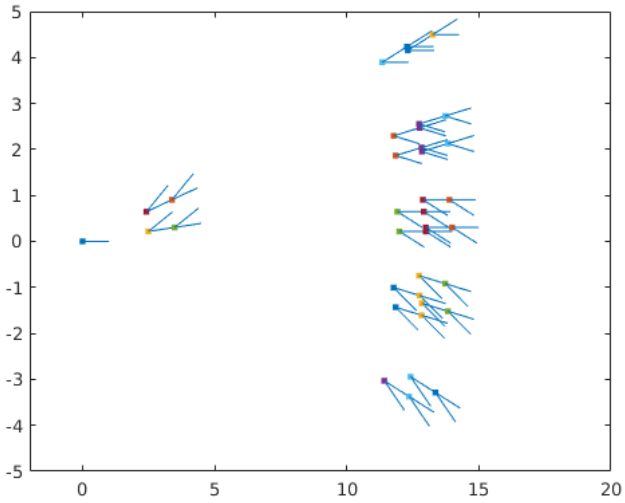

Figure 2: Pose estimates for the first noisy measurement after two iterations.

Figure 2 shows the resulting plot, where each position also includes the robot's heading. It can be seen that the plot starts to take the form of a density, with the typical banana-shape. This indicates the growing uncertainty regarding the robot's true position, coming from the error terms added in the system.

Question 2

In order to determine the bike's posterior, a two step process has to be followed: calculating the bike's pose given a specific control and determining the actual posterior.

Calculating the new bike's pose

The main element needed for determining the new pose of the bike given its initial pose, is the instantaneous center of rotation (ICR): it defines the center of the circle on which the back wheel of the bike will travel, allowing the calculation of the new pose given a steering angle and velocity.

First, to determine the ICR, the coordinate of the back wheel is needed and it can be easily calculated using basic trigonometry:

$$x_{back} = x_{front} - l \cdot cos(\theta)$$
$$y_{back} = y_{front} - l \cdot sin(\theta)$$

where x_{back} and y_{back} are the coordinates of the back wheel, x_{front} and y_{front} are the coordinates of the front wheel, l is the length of the bike frame and θ defines the angular orientation.

Having determined the rear wheel location, the next step is to calculate the turning radius of the back wheel, shown as R_I in belows Figure.

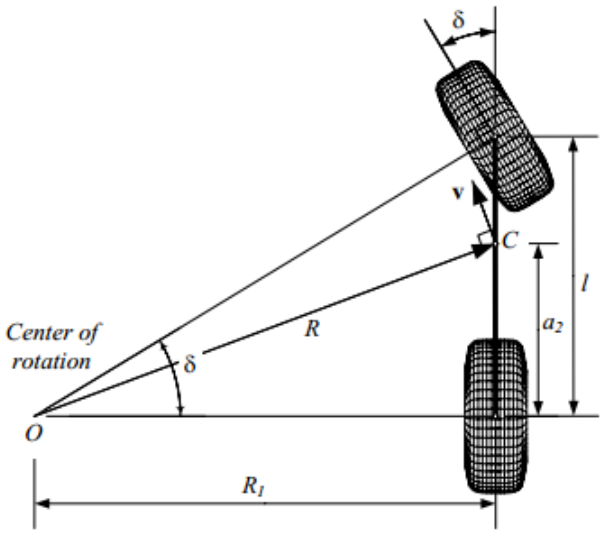

Figure 3: Instantaneous center of rotation of a bike.

The value for R_I can be determined as

$$R_I = l \cdot cot(\alpha)$$

where α is the steering angle of the bike (in figure shown as δ). Now it's possible to calculate the coordinates of the center of rotation, as:

$$x_{ICR} = x_{back} - R_I \cdot sin(\theta)$$

$$y_{ICR} = y_{back} + R_I \cdot cos(\theta)$$

The next step is to calculate the length of the arc the bike would travel at the given speed: this can be easily determined as

$$distance = v \cdot \Delta t$$

The traveled distance is needed in order to calculate the traveled angle of the bike from the starting point:

$$\gamma = \frac{distance}{R_I}$$

Note that γ is expressed in radians, and thus it needs to be converted into degrees. With all the above data, the final element needed to correctly locate the bike after its movement is the angle from the horizontal of the circle to the rear bike wheel β, used to calculate the traveled angle from the horizontal:

$$\beta = atan2(y_{back} - y_{ICR}, x_{back} - x_{ICR})$$

As before, this angle is expressed in radians and needs to be converted into degrees. The traveled angle from the horizontal is then simply

$$\gamma_{hor} = \beta + \gamma$$

The new location of the bike can now be calculated, by reversing the procedure (back wheel first then orientation and finally front wheel).

$$x_{back}^{new} = x_{ICR} + R_I \cdot cos(\gamma_{hor})$$
$$y_{back}^{new} = y_{ICR} + R_I \cdot sin(\gamma_{hor})$$

$$\theta^{new} = \theta + \gamma_{hor}$$

$$x_{front}^{new} = x_{back}^{new} + l \cdot cos(\theta^{new})$$
$$y_{front}^{new} = y_{back}^{new} + l \cdot sin(\theta^{new})$$

the new pose is then

$$\begin{bmatrix} x_{front}^{new} \\ y_{front}^{new} \\ \theta^{new} \end{bmatrix}$$

Calculating the bike's posterior

An important detail to notice is that now the control is not exact: both angle and velocity have a Gaussian noise added, removing full certainty from the system. This results in a major problem arising when calculating the posterior distribution, coming from the trigonometric calculations exposed above.

While calculating the position of the back wheel is straightforward, determining the length of the ICR radius (R_I) poses a much more difficult challenge, as a cotangent of α is required: α needs to be modeled as a normal distribution, and the cotangent operator requires both cosine and sine of an angle to be defined. This results in a division of two normal distributions, which complicates the problem of finding the posterior by a considerable amount. As this model would be used by a robot of probably limited computing power, determining the exact posterior of this system would be unfeasible.

An alternative way to solve this problem could be simplifying the calculations by means of, for example, a particle filter: multiple controls sampled from the state space of v and α could model with decent accuracy the posterior, while avoiding complex calculation and high computing power.

Question 3

The sampling process for this exercise revolves around generating a control state for each sample by adding random Gaussian noise to both wheel angle and velocity. For each particle the resulting pose is determined and plotted.

Every plot consists of 200 sampled poses, along with the original starting pose $(0,\ 0,\ 0)^T$.

Problem 1

$\alpha = 25, v = 20cm/s$

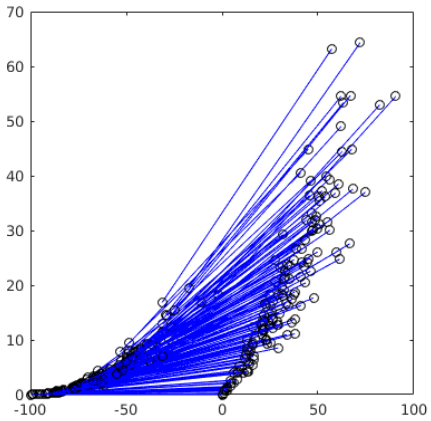

Figure 4: Sampled poses for problem 1

The poses correctly drift upwards as the front wheel is tilted left.

Problem 2

$\alpha = -25, v = 20cm/s$

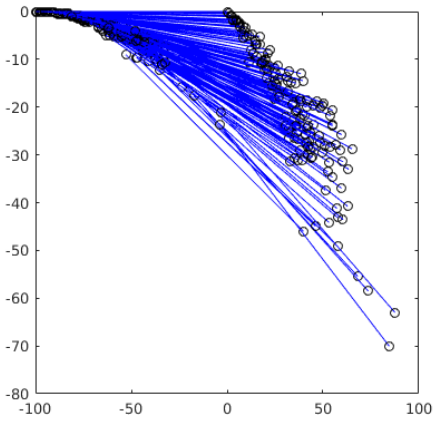

Figure 5: Sampled poses for problem 2

In this case the poses drift downward as the front wheel is turned right.

Problem 3

$\alpha = 25, v = 90cm/s$

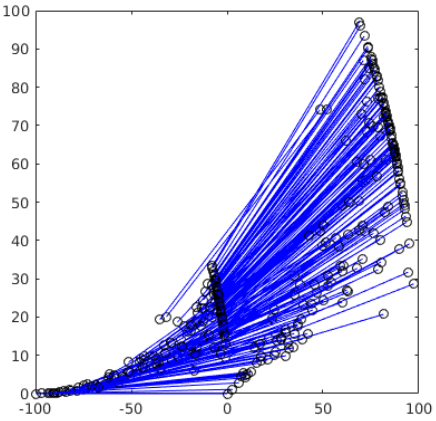

Figure 6: Sampled poses for problem 3

This problem is more peculiar, as the velocity is close to the clipping point of $100cm/s$, thus creating an edge in the right side of the plot.

Problem 4

$\alpha = 80, v = 10cm/s$

In this problem α has a very high value, generating a circle of counterclockwise motion. As the velocity is very limited, most of the poses lie in half of the circle.

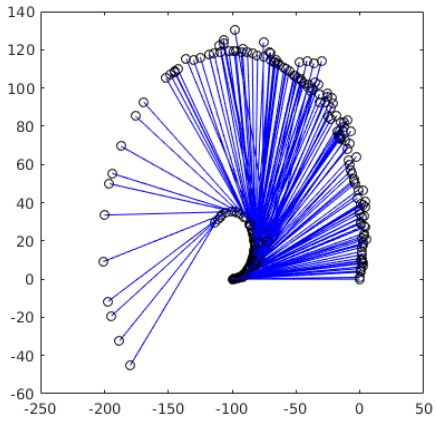

Figure 7: Sampled poses for problem 4

Problem 5

$\alpha = 85, v = 90cm/s$

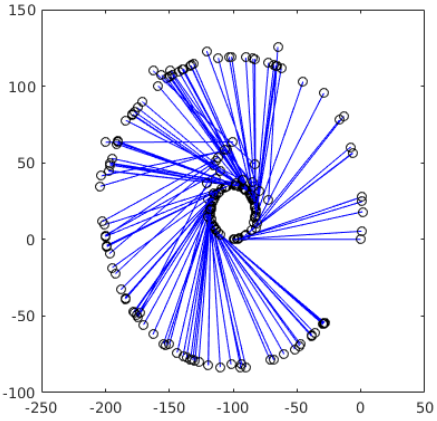

Figure 8: Sampled poses for problem 5

As before, the generated poses lie on a circle, but the higher velocity populates all the circumference. To be noted that no sample managed to do a full rotation, as the circumference is not small enough and the velocity is almost always clipped to $100cm/s$.

Matlab code

Question 1

```
% question a)
% initial pose
initial_pose = [0; 0; 0];

% measurements
measurement1 = [10; 3; 10];
measurement2 = [-20; 10; -10];

pose_1 = calculate_true_position(initial_pose, measurement1)
pose_2 = calculate_true_position(pose_1, measurement2)


% question b)
% movement errors
errors = [5; 0.5; 10];

% calculate all possible errors combinations
combined_errors = calculate_errors_combinations(errors);
combinations_number = size(combined_errors, 2);

new_poses1 = zeros(3, combinations_number);
% try every combination
for i=1:combinations_number
    new_poses1(:, i) = calculate_true_position(initial_pose, measurement1 + ...
        combined_errors(:, i));
end

% plot all the calculated poses
figure();
plot_pose(initial_pose);

for i=1:size(new_poses1, 2)
    plot_pose(new_poses1(:, i));
end

xlim([-1, 5])
ylim([-1, 2])


% question c)
new_poses2 = zeros(3, combinations_number * combinations_number);
for p=1:combinations_number
    for i=1:combinations_number
        new_poses2(:, (p-1)*combinations_number + i) = ...
            calculate_true_position(new_poses1(:, p), measurement2 + combined_errors(:, i));
    end
end

% plot the old poses
figure();
plot_pose(initial_pose);
for i=1:size(new_poses1, 2)
    plot_pose(new_poses1(:, i));
end

% plot the new poses
for i=1:size(new_poses2, 2)
    plot_pose(new_poses2(:, i));
end

xlim([-2, 20])
```

```
ylim([-5, 5])


%% support functions
% calculate the true robot's position given a pose and a measurement (5.40)
function [new_pose] = calculate_true_position(initial_pose, measurement)
    x = initial_pose(1) + measurement(2) * cosd(initial_pose(3) + measurement(1));
    y = initial_pose(2) + measurement(2) * sind(initial_pose(3) + measurement(1));
    theta = initial_pose(3) + measurement(1) + measurement(3);
    new_pose = [x; y; theta];
end

% generate all possible combinations of errors
function [combined_errors] = calculate_errors_combinations(errors)
    combinations = dec2bin(0:2^3-1);

    combined_errors = repmat(errors, 1, size(combinations, 1));
    for i=1:size(combinations, 1)
        for j=1:3
            if combinations(i, j) == '0'
                combined_errors(j, i) = -combined_errors(j, i);
            end
        end
    end
end

% plot a robot pose
function [] = plot_pose(pose)
    plot(pose(1), pose(2), '.', 'MarkerSize', 10);
    line([pose(1), pose(1) + cosd(pose(3))], [pose(2), pose(2) + sind(pose(3))]);
    hold on;
end
```

Question 3

```
% initial data
initial_pose = [0, 0, 0];
l = 100;
d = 80;
Δ_t = 1;

% variances
variance_alpha = 25;
variance_v = 50;

% samples
samples = 200;

% controls
controls = [20, 25; 20 -25; 90, 25; 10, 80; 90, 85];

% draw samples for each control
for c=1:size(controls, 1)
    figure();
    control = controls(c, :);
    plot_pose(initial_pose, l);

    % sample poses
    sampled_poses = sample_poses(initial_pose, control, Δ_t, l, variance_alpha, ...
        variance_v, samples);

    % plot poses
    for i=1:samples
        plot_pose(sampled_poses(:, i), l);
```

```
    end
    pbaspect([1, 1, 1])
end


%% support functions
% generate an output pose given starting pose and control
function [new_pose] = predict_pose(initial_pose, control, Δ_t, l)
    x_front = initial_pose(1);
    y_front = initial_pose(2);
    theta = initial_pose(3);
    v = control(1);
    alpha = control(2);

    % icr radius
    rotation_radius = l * cotd(alpha);

    % back wheel position
    x_back = x_front - l * cosd(theta);
    y_back = y_front - l * sind(theta);

    % icr position
    x_icr = x_back - rotation_radius * sind(theta);
    y_icr = y_back + rotation_radius * cosd(theta);

    % arc length
    distance_travelled = v * Δ_t;

    % angle length
    angle_travelled = rad2deg(distance_travelled / abs(rotation_radius));

    if alpha < 0
        angle_travelled = -angle_travelled;
    end

    % angle from the horizontal
    beta = atan2d(y_back - y_icr, x_back - x_icr);
    angle_travelled_horizontal = beta + angle_travelled;

    % new back wheel position
    new_x_back = x_icr + abs(rotation_radius) * cosd(angle_travelled_horizontal);
    new_y_back = y_icr + abs(rotation_radius) * sind(angle_travelled_horizontal);

    % new orientation
    new_theta = theta + angle_travelled;

    % new front wheel position
    new_x_front = new_x_back + l * cosd(new_theta);
    new_y_front = new_y_back + l * sind(new_theta);

    % new final pose
    new_pose = [new_x_front, new_y_front, new_theta];
end

% draw samples given input data
function [sampled_poses] = sample_poses(initial_pose, control, Δ_t, l, variance_alpha, ...
    variance_v, samples)
    sampled_poses = zeros(3, samples);

    for i=1:samples
        % generate a noisy control state
        noisy_control(1) = normrnd(control(1), sqrt(variance_v * abs(control(1))));
        noisy_control(2) = normrnd(control(2), sqrt(variance_alpha));

        % clip the generated control
        noisy_control = check_limits(noisy_control);

        % predict output pose
```

```
96          sampled_poses(:, i) = predict_pose(initial_pose, noisy_control, Δ_t, l);
97      end
98  end
99
100 % plot bike
101 function [] = plot_pose(pose, l)
102     x_back = pose(1) - l * cosd(pose(3));
103     y_back = pose(2) - l * sind(pose(3));
104
105     plot([pose(1), x_back], [pose(2), y_back], 'blue');
106     hold on;
107     scatter([pose(1), x_back], [pose(2), y_back], 'black');
108 end
109
110 % clip a control if over thresholds
111 function [result] = check_limits(noisy_control)
112     result = noisy_control;
113
114     % angle
115     if noisy_control(2) > 80
116         result(2) = 80;
117     elseif noisy_control(2) < -80
118         result(2) = -80;
119     end
120
121     % velocity
122     if noisy_control(1) < 0
123         result(1) = 0;
124     elseif noisy_control(1) > 100
125         result(1) = 100;
126     end
127 end
```