1 Hausübung 2

1.1 Angaben

Unsicherheitsfaktoren: $k_r = k_l = 0.001$
Radabstand: d = 20cm
Meine Angaben (Aufgabe 8):
Startposition: x = 0, y = 0, $\theta = 60°$
Pfad: 5 Schritte mit jeweils 15 cm vorwärts, ein Schritt mit Drehung um $\pi/2$, dann 5 Schritte mit jeweils 15 cm vorwärts.

1.2 Code

Sämtlicher Code befindet sich auch auf meinem git unter:
https://github.com/MaRu999/autonome_Systeme/tree/master/h2/
Funktion für die Berechnung von G_s:

```
% function for calculating the Jakobi matrix of the drive, based on the skriptum, page 53
% input parameters:
% delta_s = distance travelled in step
% delta_angle = difference in angle for step
% angle = current angle of robot
% d = wheel distance
% return values:
% jakobi = the Jakobi matrix for the drive
function jakobi = Jakobi_s(delta_s, delta_angle, angle, d)
    % variable for holding the calculation
    % variable is current angle * difference in angle/2
    val = angle + (delta_angle/2);
    % variable for holding the calculation
    % variable is distance travelled / 2 * wheel distance
    var = delta_s/(2*d);
    jakobi = [
    % 1/2 * cosinus of val - var * sinus of val, 1/2 * cosinus of val + var * sinus of val
    (0.5*cos(val) - var*sin(val)), (0.5*cos(val) + var*sin(val));
    % 1/2 * sinus of val + var * cosinus of val, 1/2 * sinus of val - var * cosinus of val
    (0.5*sin(val) + var*cos(val)), (0.5*sin(val) - var*cos(val));
    % 1/wheel distance, -1/wheel distance
    (1/d), (-1/d)
    ];
```

Funktion für die Berechnung von G_p

```
% function for calculating the Jakobi matrix of the pose, based on skriptum, page 52
% input parameters:
% delta_s = distance travelled in step
% delta_angle = difference in angle for step (meaning rotation)
% angle = current angle of the robot
% return values:
% jakobi = the calculated Jakobi matrix for the pose
function jakobi = Jakobi_p(delta_s, delta_angle, angle)
    % variable for holding result of expression so it does not need to be typed every time
    % variable is current angle of robot + difference in angle (rotation of robot for step)/2
    val = angle + (delta_angle/2);
    jakobi = [
    % 1, 0, -distance travelled for step * sinus of val
    1, 0, (-delta_s * sin(val));
    % 0, 1, distance travelled for step * cosinus of val
    0, 1, (delta_s * cos(val));
    0, 0, 1
    ];
```

Die mathematische Formel für die Berechnung von Δs_r und Δs_l steht nicht explizit im Skriptum, sie wurde selbst hergeleitet: Von den Folien stammen die Vorbedingungen:

$$\begin{aligned} \Delta s &= \frac{\Delta s_r + \Delta s_l}{2} \\ \Delta\theta &= \frac{\Delta s_r - \Delta s_l}{d} \end{aligned} \tag{1}$$

Wir multiplizieren jetzt die erste Zeile mit 2 und die zweite Zeile mit d:

$$\begin{aligned} 2\Delta s &= \Delta s_r + \Delta s_l \\ \Delta\theta d &= \Delta s_r - \Delta s_l \end{aligned} \tag{2}$$

Nun addieren wir die erste Zeil zur zweiten Zeile:

$$\Delta\theta d + 2\Delta s = 2\Delta s_r \tag{3}$$

Wir dividieren durch zwei:

$$\frac{\Delta\theta d + 2\Delta s}{2} = \Delta s_r \tag{4}$$

Nun haben wir eine Formel für Δs_r. Wir formen nun $2\Delta s = \Delta s_r + \Delta s_l$ nach Δs_l um:

$$\Delta s_l = 2\Delta s - \Delta s_r \tag{5}$$

Diese Formeln benutzen wir im Code, der folgt: Funktion für die Berechnung der Δs der Räder.

```
% function for calculating the deltas for the wheels, meaning the distance the left and right wheel travel in the step
% input paramters:
% delta_s = distance traveled in the step
% d = wheel distance
% delta_angle = difference in angle (rotation for this step)
% return values:
% delta_sr = distance travelled in the step for the right wheel
% delta_sl = distance travelled in the step for the left wheel
function [delta_sr, delta_sl] = Get_wheel_deltas(delta_s, d, delta_angle)
    % calculation based on skriptum, page 51
    % distance travelled in step for right wheel is (2 times total distance travelled in step + wheel distance * rotation divided) by 2
    delta_sr = ((2*delta_s) + (d * delta_angle))/2;
    % distance travelled in step for left wheel is 2 times total distance travelled - distance travelled for right wheel
    delta_sl = (2*delta_s) - delta_sr;
```

Funktion für die Berechnung der Kovarianzmatrix der Räder:

```
% function that calculates the covariance matrix of the drive, as shown in skriptum on page 52
% input parameters:
% k_r = uncertainty factor for covariance matrix for righ wheel of drive
% delta_sr = distance driven for step for right wheel
% k_l = uncertainty factor for covariance matrix for left wheel of drive
% delta_sl = distance driven for step for left wheel
% return values:
% covar = covariance matrix for the drive
function covar = Covariance_drive(k_r, delta_sr, k_l, delta_sl)
    covar = [
    % k_r * amount (Betrag) vector delta_sr, 0
    (k_r * norm(delta_sr)), 0;
    % 0, k_l * amount (Betrag) vector delta_sl
    0, (k_l * norm(delta_sl))
    ];
```

Funktion für die Berechnung der nächsten Kovarianzmatrix:

```
% function that calculates the next covariance matrix
% input parameters:
% jakobi_p = Jakobi matrix for pose
% covar_p = current covariance matrix
% jakobi_s = Jakobi matrix for drive
% covar_s = covariance matrix for drive
% return values:
```

```
% next = the next covariance matrix
function next = Covar_next(jakobi_p, covar_p, jakobi_s, covar_s)
    % the next covariance matrix is Jakobi pose * current covariance matrix * transposed Jakobi pose
    % + Jakobi drive * covariance matrix drive * transposed Jakobi drive
    next = (jakobi_p * covar_p * jakobi_p') + (jakobi_s * covar_s * jakobi_s');
```

Helferfunktion, die anhand der Eingabewerte alle nötigen Vorberechnungen (Jakobimatrizen, etc.) mit den entsprechenden Funktionen ausführt und dann die nächste Kovarianzmatrix mit Covar_next berechnet:

```
% helper function that collects the different Function calls needed for the calculation of the next covariance matrix
% input parameters:
% delta_s = the size of the step the robot takes (the distance it travels)
% delta_angle = difference in angle (how much the robot rotates for this step)
% angle = angle of the robot (third value of the current pose vector)
% d = wheel distance
% k_r = uncertainty factor for covariance matrix for righ wheel of drive
% k_l = uncertainty factor for covariance matrix for left wheel of drive
% covar_p = current covariance matrix
% return values:
% next_covar = the next covariance matrix
function next_covar = Calc_next_covar(delta_s, delta_angle, angle, d, k_r, k_l, covar_p)
    % calculate the distances the left and right wheel travel for this step
    [delta_sr, delta_sl] = Get_wheel_deltas(delta_s, d, delta_angle);
    % calculate the jakobi matrix p (for the pose)
    jakobi_p = Jakobi_p(delta_s, delta_angle, angle);
    % calculate the jakobi matrix s (for the drive)
    jakobi_s = Jakobi_s(delta_s, delta_angle, angle, d);
    % calculate the covariance matrix for the dive
    covar_s = Covariance_drive(k_r, delta_sr, k_l, delta_sl);
    % calculate the next covariance matrix
    next_covar = Covar_next(jakobi_p, covar_p, jakobi_s, covar_s);
```

Funktion, die die nächste Pose des Roboters berechnet:

```
% function for calculating the robot's new pose
```

```
% input parameters:
% old_pose = the current pose of the robot
% delta_s = the distance the robot travels
% delta_angle = the angle at which the robot travels
% return values:
% new_pose = the new pose of the robot
function new_pose = Calc_new_pose(old_pose, delta_s,
    delta_angle)
    % this variable is only here to make the code more readable
     and to
    % not have to write the whole expressions every time
    % value is the value at index three of the current pose (
    meaning the angle) + the difference in angle divided by two
    val = old_pose(3) + (delta_angle/2);
    % change vector, same as from the skriptum, page 51f
    change_vector = [
    % step size multiplied with cosinus of val
    delta_s * cos(val);
    % step size multiplied with sinus of val
    delta_s * sin(val);
    % difference in angle
    delta_angle;
    ]
    % add the change vector to the old pose
    new_pose = old_pose + change_vector;
```

Funktion, die bei gleichbleibenden Eingabewerten eine beliebige Anzahl Schritte hintereinander berechnet und plottet:

```
% function for calculating and drawing a given number of steps
% input parameters:
% start = the current pose of the robot
% delta_s = distance to travel in the step
% delta_angle = differnence in angle, how much to rotate in
    step
% d = wheel distance
% k_r = uncertainty factor for covariance matrix for righ wheel
     of drive
% k_l = uncertainty factor for covariance matrix for left wheel
     of drive
% covariance_start = the current covariance matrix for the
    robot
% num_loops = the number of steps to perform
% return values:
% start = the new pose of the robot
% covariance_start = the new covariance matrix of the robot
function [start, covariance_start] = Draw_Loop(start, delta_s,
    delta_angle, d, k_r, k_l, covariance_start, num_loops)
    % iteration over the wanted number of steps
    for i=1:num_loops
```

```
        % get the current angle of the robot
        angle = start(3);
        % calculate the next pose
        res = Calc_new_pose(start, delta_s, delta_angle);
        % calcualte the next covariance matrix
        covar = Calc_next_covar(delta_s, delta_angle, angle, d,
    k_r, k_l, covariance_start);
        % get the part of the covariance matrix neede to draw
   the ellipse
        covar_ell = covar(1:2,1:2);
        % convert the extracted part pf the covariance matrix
   to an ellipse
        elli = cov2ellipse(covar_ell);
        % move the x value of the ellipse to the x value of the
    pose
        elli(1,1) = res(1,1);
        % move the y value of the ellipse to the y value of the
    pose
        elli(1,2) = res(2,1);
        % plot the calculated pose
        plot(res(1,:), res(2,:), 'r+');
        % draw the ellipse
        drawEllipse(elli);
        % set start to calculated new pose
        start = res;
        % set covariance to calculated new covariance matrix
        covariance_start = covar;
    end
```

Skript, das die Aufgabe 8 mit hilfe der anderen Funktionen berechnet:

```
pkg load geometry;
pkg load matgeom;
% script for task 8

% distance to travel in step
delta_s = 0.15;
% angle for most steps
delta_angle = 0;
% wheel distance
d = 0.2;
% uncertainty factor for covariance matrix for righ wheel of
   drive
k_r = 0.001;
% uncertainty factor for covariance matrix for left wheel of
   drive (same as right wheel in this example)
k_l = k_r;
% starting covariance matrix of 3x3 filled with zeros
covariance_start = zeros(3,3);
% starting pose of robot at x = 0, y = 0, theta = 60 degrees
```

```
start = [
0;
0;
% since we are using cos and sin, we need rads, so transform from degree to rad
deg2rad(60)
]

% open new figure window
figure();
% hold figure window (keep drawing in same window)
hold on
% plot the starting point (no uncertainty, so no ellipse needed)
plot(start(1,1), start(2,1), 'r+');
% set label for x axis
xlabel("x");
% set label for y axis
ylabel("y");
% take the first five steps with the values set above
[start, covariance_start] = Draw_Loop(start, delta_s, delta_angle, d, k_r, k_l, covariance_start, 5);
% take one step that travels no distance, but rotates the robot by pi/2
[start, covariance_start] = Draw_Loop(start, 0, pi/2, d, k_r, k_l, covariance_start, 1);
% take five more steps with values set at the beginning of the script
[start, covariance_start] = Draw_Loop(start, delta_s, delta_angle, d, k_r, k_l, covariance_start, 5);
% print image to file
print -dpng -r300 task8v2.png;
% stop holding figure
hold off
```

1.3 Resultat

Hier noch das anhand dieses Codes geplottete Bild:

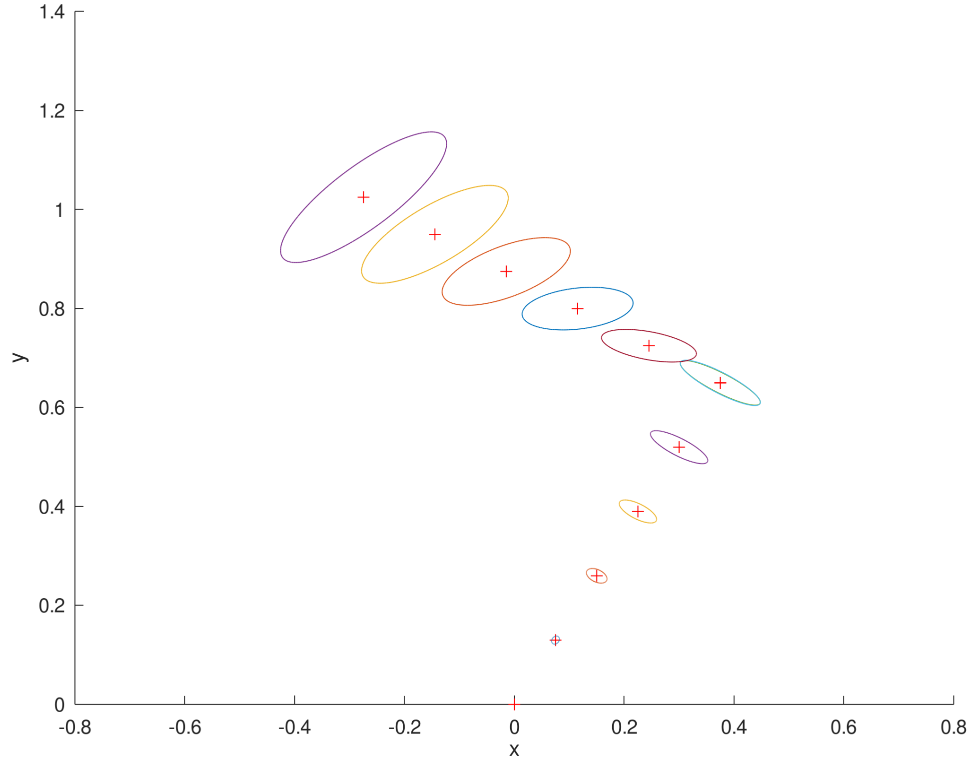

Abbildung 1: Mit Code erstellter Plot zu Aufgabe 8