Simultaneous Localization and Mapping (SLAM)

Submitted to:

Dr. Fabien Bonardi

By

Fatma Mohamed

Contents

List of Figures

Chapter 1

Introduction

The term SLAM is as stated an acronym for Simultaneous Localization And Mapping. It was originally developed by Hugh Durrant-Whyte and John J. Leonard based on earlier work by Smith, Self and Cheeseman . Durrant-Whyte and Leonard originally termed it SMAL but it was later changed to give a better impact. SLAM is concerned with the problem of building a map of an unknown environment by a mobile robot while at the same time navigating the environment using the map. SLAM consists of multiple parts; Landmark extraction, data association, state estimation, state update and landmark update. SLAM is applicable for both 2D and 3D motion. We will only be considering 2D motion.

SLAM involves a moving agent (for example a robot), which embarks at least one sensor able to gather information about its surroundings (a camera, a laser scanner, a sonar: these are called exteroceptive sensors). Optionally, the moving agent can incorporate other sensors to measure its own movement (wheel encoders, accelerometers, gyrometers: these are known as proprioceptive sensors). The minimal SLAM system consists of one moving exteroceptive sensor (for example, a camera in your hand) connected to a computer.

So, Basically SLAM is computing the robot's poses and the map of the environment at the same time. The main steps for EKF SLAM are;

1. The robot moves, reaching a new point of view of the scene. Due to noise and errors, this motion increases the uncertainty on the robot's localization. An automated solution requires a mathematical model for this motion. Which is called the motion model (move.m)

- The robot discovers landmarks in the environment, which need to be incorporated to the map. Because of errors in the exteroceptive sensors, the location of these landmarks will be uncertain. Moreover, as the robot location is already uncertain, these two uncertainties need to be properly composed. An automated solution requires a mathematical model to determine the position of the landmarks in the scene from the data obtained by the sensors. which is called the backProject model (backProject.m)

1. The robot observes landmarks that had been previously mapped, and uses them to correct both its self-localization and the localization of all landmarks in space. In this case, therefore, both localization and landmarks uncertainties decrease. An automated solution requires a mathematical model to predict the values of the measurement from the predicted landmark location and the robot localization. Which is called the scan mdel (scan.m)

Chapter 2

The SLAM problem

2.1 Problem defintion

The model mainly includes three equations:

- Motion equation: it will increase the uncertainty of robot positioning .
- The equation for initializing road signs based on observation: It initializes new state quantities based on observation information.
- The projection equation of the road sign state to the observation: According to the observation information, the state is updated, corrected, and the uncertainty is reduced

Some remarks regarding the SLAM problem;

- Robot path and map are both unknown and estimated at the same time.
- Robot path error estimation correlates with errors in the map.
- In real environments, mapping between observations and landmarks is unknown.
- Making a wrong data associations can have very bad consequences for map and path estimations.
- Pose error correlates data associations errors too

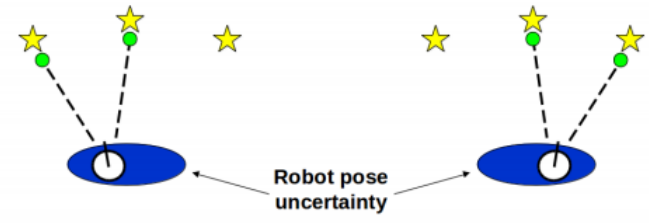


Figure 2.1: Robot pose uncertainty

so, what are the givens and the wanted for the SLAM problem ?

- **Given** : The robots controls u_T and the observations Z_T

$$u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\} \quad , \quad z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$$

- **Wanted** : Map of the environment m and Path of the robot x_T

$$x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\}$$

2.2 Complexity of the problem

- Cubic complexity depends only on the measurement dimensionality.
- Cost per step: dominated by the number of landmarks.
- Memory consumption.
- The EKF becomes computationally intractable for large maps.

Chapter 3

EKF SLAM Algorithm

3.1 Extended Kalman filter Algorithm

Starting with the Extended Kalman Filter algorithm in order to apply it to the SLAM problem.

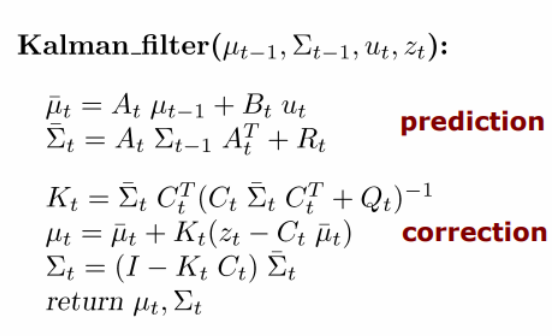


Figure 3.1: EKF Algorithm

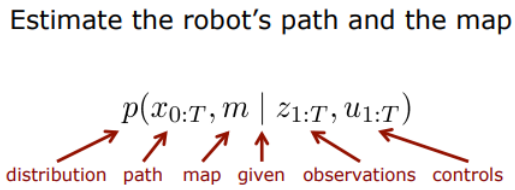
Where μ_{t-1}represents the mean of the previous estimation, Σ_{t-1} represents the covarience matrix of the previous estimation, $u_t$represents the control input of the current time step, and $z_t$represents the observation of the property we need to estimate in the current time step.

Where the First two steps are considered the Prediction steps and the last four steps are considered the Correction step.

3.2 EKF Applied to SLAM problem

As mentioned before, In EKF-SLAM, the map is a stack of landmarks states, modeled by Gaussian variables. This map is updated thanks to the process of prediction (motion and sensor model combination) and correction (observation of the landmarks with the sensors are matched with previously mapped landmarks). For exploration scenarios, landmarks initialization has to be made : newly discovered landmarks are added to the map.

In the point of view of the Gaussian filters (i.e. EKF) the model of SLAM is presented as follows:



For the graphical model,

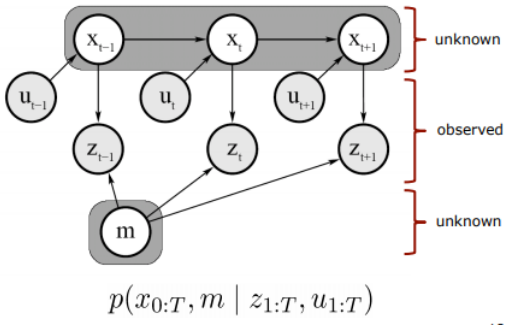


Figure 3.2: Graphical model of EKF applied to SLAM problem

Robot motion

The state is a vector with robot pose and landmark positions :

$$x = \begin{bmatrix} R \\ M \end{bmatrix} = \begin{bmatrix} R \\ L_1 \\ \vdots \\ L_n \end{bmatrix}$$

In EKF, this state is modeled by a Gaussian variable, using mean and covariance matrix :

$$\tilde{x} = \begin{bmatrix} \tilde{R} \\ \tilde{M} \end{bmatrix}$$

$$P = \begin{bmatrix} P_{RR} & P_{RM} \\ P_{MR} & P_{MM} \end{bmatrix}$$

Initial state (2D-SLAM) can be defined such that :

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$P = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In regular EKF, time-update function is :

$$x_{t+1} \leftarrow f(x_t, u, n)$$

EKF prediction step can be written as :

$$\tilde{x}_{t+1} \leftarrow f(\tilde{x}_t, u, 0)$$

$$P_{t+1} \leftarrow F_x P_t F_x^T + F_n N F_n^T$$

with the Jacobian matrices :

$$F_x = \frac{\partial f(\tilde{x}, u)}{\partial x} \text{ and } F_x = \frac{\partial f(\tilde{x}, u)}{\partial n}$$

where N is covariance matrix of the noise n
In SLAM, only the robot state is time-variant :

$$R_{t+1} \leftarrow f_R(R_t, u, n) \text{ (motion model)}$$

$$M_{t+1} \leftarrow M_t$$

and

$$F_x = \begin{bmatrix} \frac{\partial f_R}{\partial R} & 0 \\ 0 & I \end{bmatrix}$$

$$F_x = \begin{bmatrix} \frac{\partial f_R}{\partial n} \\ 0 \end{bmatrix}$$

and because of trivial operations, we have :

$$\tilde{R}_{t+1} \leftarrow f_R(\tilde{R}, u, 0)$$

$$P_{RR} \leftarrow \frac{\partial f_R}{\partial R} P_{RR} \frac{\partial f_R}{\partial R}^T + \frac{\partial f_R}{\partial n} N \frac{\partial f_R}{\partial n}^T$$

$$P_{RM} \leftarrow \frac{\partial f_R}{\partial R} P_{RM}$$

$$P_{MR} \leftarrow P_{RM}^T$$

3.2.1 Observation of Mapped Landmarks

In EKF, the generic observation function is :

$$y = h(x) + v$$

where y is the noisy measurement, x the full state, h() observation function and v the measurement noise. EKF correction step is defined as :

$$\tilde{z} = y - h(\tilde{x})$$

$$Z = H_x P H_x^T + R$$

$$K = P H_x^T Z^{-1}$$

$$\tilde{x} \leftarrow \tilde{x} + K\tilde{z}$$

$$P \leftarrow P - KZK^T$$

with the Jacobian $H_x = \frac{\partial h(x)}{\partial x}$ and R covariance matrix of the measurement noise.
Assuming that landmark i is observed, we have the individual observation function :

$$y_i = h_i(R, S, L_i) + v$$

which does not depend on any other landmark $\rightarrow$Jacobian H_x is also sparse :

$$H_x = \begin{bmatrix} H_R & 0 & \dots & 0 & H_{L_i} & 0 & \dots & 0 \end{bmatrix}$$

with $H_R = \frac{\partial h_i(R,S,L_i)}{\partial R}$ and$H_{L_i} = \frac{\partial h_i(R,S,L_i)}{\partial L_i}$

The set of equation becomes :

$$\tilde{z} = y_i - h_i(\tilde{R}, S, \tilde{L}_i)$$

$$Z = \begin{bmatrix} H_R & H_{L_i} \end{bmatrix} \begin{bmatrix} P_{RR} & P_{RL_i} \\ P_{L_iR} & P_{L_iL-i} \end{bmatrix} \begin{bmatrix} H_R^T \\ H_{L_i} \end{bmatrix} + R$$

$$K = \begin{bmatrix} P_{RR} & P_{RL_i} \\ P_{MR} & P_{ML-i} \end{bmatrix} \begin{bmatrix} H_R^T \\ H_{L_i} \end{bmatrix} Z^{-1}$$

$$\tilde{x} \leftarrow \tilde{x} + K\tilde{z}$$

$$P \leftarrow P - KZK^T$$

3.2.2 Landmarks Intialization

Landmark initialization happens when the robot discovers landmarks that are not yet in the map :

$$L_{n_1} = g(R, S, y_{n+1})$$

We deduce landmark's mean anbd Jacobians :

$$\tilde{L}_{n_1} = g(\tilde{R}, S, y_{n+1})$$

$$G_R = \frac{\partial g(\tilde{R}, S, y_{n+1})}{\partial R}$$

$$G_{y_{n+1}} = \frac{\partial g(\tilde{R}, S, y_{n+1})}{\partial y_{n+1}}$$

We can then compute covariance matrices :

$$P_{LL} = G_R P_{RR} G_R^T + G_{y_{n+1}} R G_{y_{n+1}}^T$$

$$P_{Lx} = G_R P_{Rx} = G_R [P_{RR}\ P_{RM}]$$

We can then append these results to the state mean and covariances matrix :

$$\tilde{x} \leftarrow \begin{bmatrix} \tilde{x} \\ \tilde{L}_{n+1} \end{bmatrix}$$

$$P \leftarrow \begin{bmatrix} P & P_{Lx}^T \\ P_{Lx} & P_{LL} \end{bmatrix}$$

Chapter 4

EKF SLAM Simulation

4.1 Sensor mode

We assumed the usage of a 360 A high-degree radar that can detect and find landmark points and their types within a certain range around them. The detection result is $(\boldsymbol{\xi}, \boldsymbol{s})$. where $(\boldsymbol{\xi})$ is the angle between the landmark point vector and the current radar coordinate system, (s) is the distance between the road marking point and the origin of the radar local coordinate system.

4.2 Motion Model and Observation

Set the $(u_1, 0)$ point in the radar local coordinate system at the current time as the position of the radar at the next time:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} u_1 \\ 0 \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} n_1 \\ 0 \end{bmatrix}$$

where its orientation θ is

$$\theta = \theta + u_2 + n_2$$

Suppose the state of the observed landmark point $\boldsymbol{i}$ is(m_1^i, m_2^i) , then its coordinates in the current radar coordinate system are:

$$\begin{bmatrix} {ladar_1}^i \\ {ladar_2}^i \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}^{-1} \begin{bmatrix} {m_1}^i \\ {m_2}^i \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix}$$

then the observation is

$$\xi = atan2\left({ladar_2}^i, {ladar_1}^i\right) + v_1$$

$$s = \sqrt{\left({ladar_1}^i\right)^2 + \left({ladar_2}^i\right)^2} + v_2$$

Chapter 5

Results

1. As shown in the figure below, the black covariance ellipse indicates that the radar has observed this landmark before, but it is not currently observed. The covariance ellipse indicates that the radar has observed it and is currently observing it

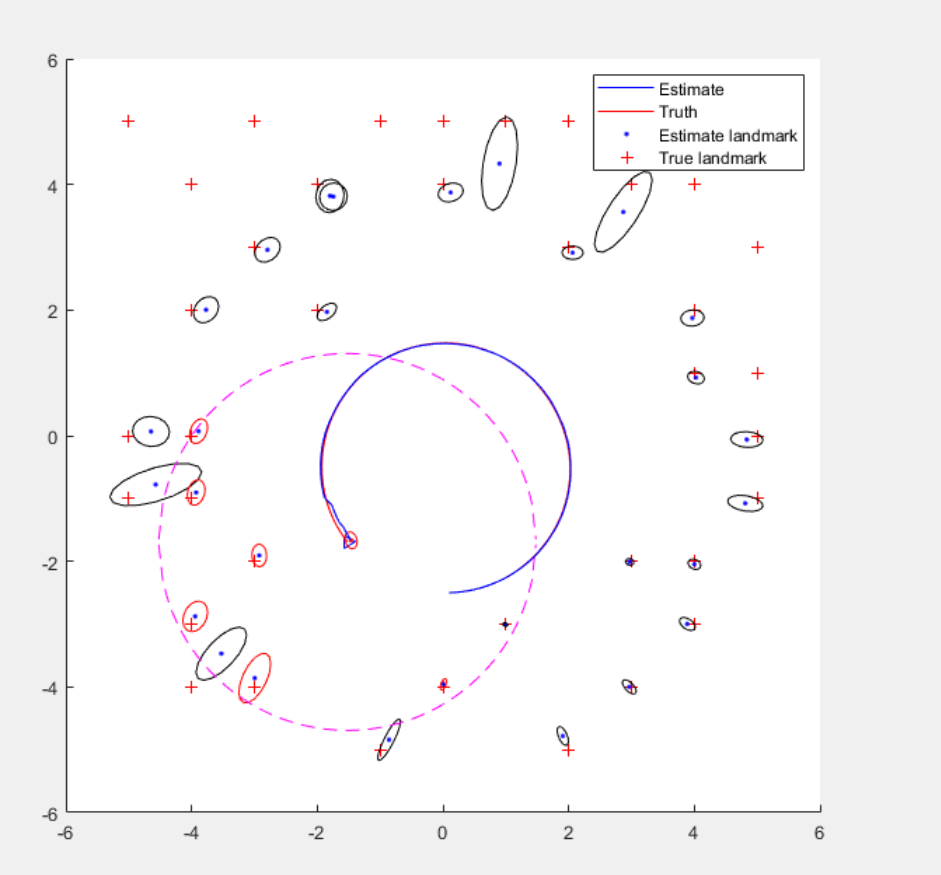


Figure 5.1: Results of different landmarks detection

This is done throught dividing the map into two parts:

- i_olds :represents the collection of old landmark points that have been observed.
- i_new represents a collection of newly discovered landmarks.

and to intialize them both before start moving the Robot, then to continously check what's inside the range of the sensor and what's not in order to mark the new landmarks differently from the old ones.

This isn't a necessary part of the EKF Slam model and can be neglected.

2. The Robot is represented by the blue Triangle. and the magenta dotted circle represents the diameter of the sensor (sensor range).

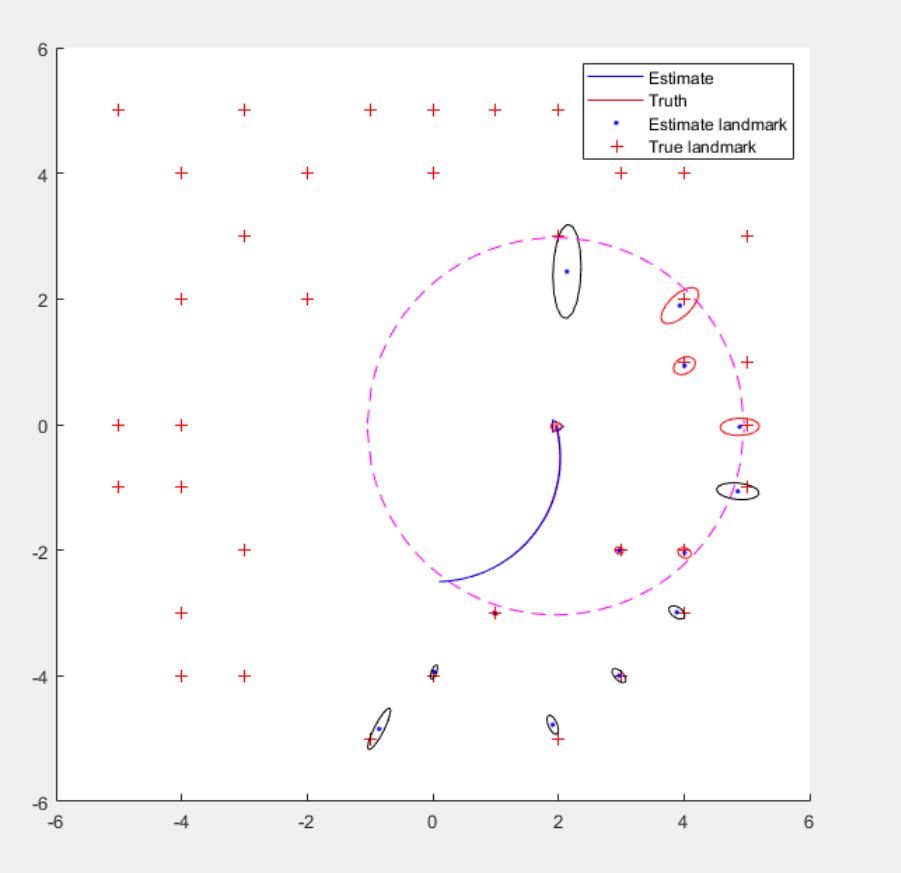


Figure 5.2: Intial moveme of the Robot

Chapter 6

References

1. Lectures Handouts.
2. http://www.probabilistic-robotics.org/
3. http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti10titsmag.pdf
4. https://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/optreadings/JulierUhlmann-UKF.pdf

Chapter 7

Appendix

7.1 Functions

```
function [p, P_r, P_y] = backProject(r, y)

if nargout == 1

    p_r = invScan(y);
    p   = fromFrame2D(r, p_r);

else

    [p_r, PR_y]    = invScan(y);
    [p, P_r, P_pr] = fromFrame2D(r, p_r);

    %
    P_y = P_pr * PR_y;

end

end

function f()

syms rx ry ra yd ya real
r = [rx;ry;ra];
y = [yd;ya];
[p, P_r, P_y] = backProject(r, y);
simplify(P_r - jacobian(p,r))
simplify(P_y - jacobian(p,y))

end
```

```
Not enough input arguments.

Error in backProject (line 10)
    [p_r, PR_y]    = invScan(y);
```

Published with MATLAB® R2020b

```
%p is the point of the global coordinate system, r is the pose of the
 local coordinate system (x, y, a)
% Projection model
function [y, Y_r, Y_p] = project(r, p)


if nargout == 1
    p_r = toFrame2D(r, p);
    y   = scan(p_r);
else

    [p_r, PR_r, PR_p] = toFrame2D(r, p);
    [y, Y_pr]   = scan(p_r);

    Y_r = Y_pr * PR_r;
    Y_p = Y_pr * PR_p;

end

end

function f()

syms px py rx ry ra real
r = [rx;ry;ra];
p = [px;py];
[y, Y_r, Y_p] = project(r, p);
simplify(Y_r - jacobian(y,r))
simplify(Y_p - jacobian(y,p))

end
```

```
Not enough input arguments.

Error in project (line 11)
    [p_r, PR_r, PR_p] = toFrame2D(r, p);
```

Published with MATLAB® R2020b

```
function [p, P_y] = invScan(y)

d = y(1);
a = y(2);

px = d * cos(a);
py = d * sin(a);

p = [px;py];

if nargout > 1

   P_y = [...
       cos(a) -d*sin(a)
       sin(a)  d*cos(a)];

end
```

```
Not enough input arguments.

Error in invScan (line 3)
d = y(1);
```

Published with MATLAB® R2020b

```
%   d = sqrt(px^2 + py^2) + rd
%   a = atan2(py, px) + ra
function [y, Y_x] = scan(x)

px = x(1);
py = x(2);

d = sqrt(px^2 + py^2);
a = atan2(py, px);
% a = atan(py/ px);
y = [d;a];

% y=atan(x)
%1/(1+x^2)
if nargout > 1

    Y_x =[...
[     px/(px^2 + py^2)^(1/2), py/(px^2 + py^2)^(1/2)]
[ -py/(px^2*(py^2/px^2 + 1)), 1/(px*(py^2/px^2 + 1))]];


end

end
```

```
Not enough input arguments.

Error in scan (line 5)
px = x(1);
```

```
function f()

syms px py vx vy real
x = [px;py;vx;vy];
y = scan(x);
Y_x = jacobian(y, x)

end
```

Published with MATLAB® R2020b

```
function [p, P_r, P_pr] = fromFrame2D(r, p_r)

t = r(1:2);
a = r(3);

R = [cos(a) -sin(a) ; sin(a) cos(a)];

p = R*p_r + t;

if nargout > 1
    px = p_r(1); %
    py = p_r(2);

    P_r = [...
        [ 1, 0, - py*cos(a) - px*sin(a)]
        [ 0, 1,   px*cos(a) - py*sin(a)]];

    P_pr = R;

end

end
```

```
Not enough input arguments.

Error in fromFrame2D (line 3)
t = r(1:2);
```

```
function f()

syms x y a px py real
r = [x y a]';
p_r = [px py]';
p = fromFrame2D(r, p_r);
P_r = jacobian(p, r)

end
```

Published with MATLAB® R2020b

```
% Motion model
% Angle only adds fixed input and noise each time
% The next moment will reach the (dx, 0)
%point in the current local coordinate system
function [ro, RO_r, RO_n] = move(r, u, n, dt)


a = r(3);
dx = u(1) + n(1);
da = u(2) + n(2);

ao = a + da;
dp = [dx;0];


if ao > pi
    ao = ao - 2*pi;
end
if ao < -pi
    ao = ao + 2*pi;
end

if nargout == 1
    to = fromFrame2D(r, dp);
else
    [to, TO_r, TO_dp] = fromFrame2D(r, dp);
    AO_a  = 1;
    AO_da = 1;

    RO_r = [TO_r ; 0 0 AO_a];
    RO_n = [TO_dp(:,1) zeros(2,1) ; 0 AO_da];
end
ro = [to;ao];

end

function f()

syms x y a dx da real
X = [x;y;a];
u = [dx;da];
[xo, XO_x, XO_u] = move(X, u, zeros(2,1));
simplify(XO_x - jacobian(xo,X))
simplify(XO_u - jacobian(xo,u))

end
```

```
Not enough input arguments.

Error in move (line 8)
a = r(3);
```

Published with MATLAB® R2020b

```
% p is the point of the global coordinate system,
%r is the pose of the local coordinate system (x, y, a)
function [p_r, PR_r, PR_p] = toFrame2D(r , p)

t = r(1:2);
a = r(3);

R = [cos(a) -sin(a) ; sin(a) cos(a)];

p_r = R' * (p - t);

if nargout > 1
    px = p(1);
    py = p(2);
    x = t(1);
    y = t(2);

    %Calculate the Jacobian of p_r to r and p
    PR_r = [...
        [ -cos(a), -sin(a),   cos(a)*(py - y) - sin(a)*(px - x)]
        [  sin(a), -cos(a), - cos(a)*(px - x) - sin(a)*(py - y)]];
    PR_p = R';

end

end
```

```
Not enough input arguments.

Error in toFrame2D (line 5)
t = r(1:2);
```

```
function f()

syms x y a px py real
r = [x y a]';
p = [px py]';
p_r = toFrame2D(r, p);
PR_r = jacobian(p_r, r)

end
```

Published with MATLAB® R2020b

7.2 The main code

The main code is too long and has been put in a seperate pdf