# Probabilistic Robotics
# Exercises on Kalman Filter
# (DRAFT)

giorgio

October 3, 2023

## Abstract

In this little document we will present four exercises on Kalman filter. They represent a reasonable sample set for part of the exam on (E)KF. In the Introduction we present in short a methodology to instantiate a filter. Subsequently we propose a bunch of exercises of increasing complexity, with the aim to highlight the typical design tricks. For completeness, we added an appendix on uniform velocity planar motion, that serves as basis for many of the transition models presented in the exercises. Read that first.
**WARNING: this document has been witten in open loop mode. It will contain typos. If you start getting a headache because something sounds odd, contact me first.**

## Contents

## 1 Introduction

To instantiate an Extended Kalman filter for a specific problem we will need toto identify the system's model. Subsequently, we can validate our design by doing further checks on the system's domains and on the dimensional feasibility of the matrices and vectors involved in the EKF machinery. This can be achieved by answering the following questions:

- **What is the state of the system?**
  The answer to this question is: all I need to know to predict the behavior of the system in the future, while forgetting about the past. Sometimes you might augment the state with variables you are not immediately interested in, but are useful for your calculations. The typical cases are when dealing with IMUs or GPS where one is interested in having either an estimate of the orientation, or of the position. Yet, to calculate the observations, or to predict the evolution of the system having only the orientation or the position of a system is not sufficient. Quentities such as add biases, velocities, time skews etc.. need to be added to the system to predict future states or to calculate the raw measurements of the physical sensors.

  So said, defining the state means: defining its domain, and chosing a parameterization for it. In the case of Kalman, we are restricted to vector like parameterizations, that might lead to wraparounds if we have orientations or other non-euclidean subspaces in our state.

- **What are the controls?**
  The controls are the quantities that influence the system evolution. In the filtering context we are not really "issuing" controls to the system, but rather observing the controls that are given to the system. In general we can refer to the controls as the *observed* input.

  To define the controls, we have to identify the domain, and choose an Euclidean parameterization for the elements of this domain.

- **What are the observations?**
  The observations are the quantities through the system's state manifests itself. Often we improperly refer to the observations as measurements, since they are related to quantities computed by some sensors whose outcome can be *completely determined* assuming a known state and a correct model.

- **How do I move to the next time epoch?**
  This concerns the transition function, that computes the next state from the previous state and control. From the transition function you can immediately write a locally linear approximation of your system's transition by computing the Jacobian matrices. To assemble the transition function, you put yourself in the conditions of *total knowledge* about the previous state and control.

- **What do I expect to see?**
  This concerns the observation function, that computes the measurement *you should get* if you knew the state. Together with the observation function, you can compute its Jacobian that models a locally linear approximation of our observation function. Having a null *row* in this Jacobian means that the corresponding measurement does not depend on the state, so either you modeled badly your system, or the sensor is mounted on another robot.

- **Are the dimensions consistent?**
  Check that the dimensions of the matrices in the game are consistent with the operations of the EKF. This is not required by the framework, but gives you a good insight on weather you modeled the problem consistently.

- **Are the domains handled correctly?**
  A big evil infesting the EKF design are the non-Euclidean domains that are dealt with as if they

were Euclidean. There are two points when we do dangerous operations on these spaces, and they are all in the mean update:

$$\mu_{t|t} = \underbrace{\mu_{t|t-1} - \mathbf{K}_t(\overbrace{\mathbf{z}_t - \mu_z}^{\text{danger 1}})}_{\text{danger 2}} \tag{1}$$

In *danger 1* we subtract two things entities are not necessarily Euclidean and might contain elements such as angles. In this case you need to normalize this quantity before evaluating the expression. In *danger 2* we sum a perturbation to the current state. If our state contains non-Euclidean components, it can assume invalid values after the sum, thus it needs to be normalized again.

- **How do I implement it? Why doesn't it work?**

  There are other aspects that I left temporarily out of this document. They are related to the initial covariance of the system $\mathbf{\Sigma}_{0|0}$, and to the measurement and control covariances $\mathbf{\Sigma}_z$ and $\mathbf{\Sigma}_u$. Also the initial mean $\mu_{0|0}$ plays an important role in the success of an implementation. As a rule of thumb, I recommend you to test the system in simulation, initialize it to the true state and feed it with noise free controls and observations. If it does not work under these ideal conditions, it will never ever work.

  Subsequently you can relax these constraints one by one, first adding noise, then perturbating the initial guess and see what happens.

  There are formal ways to answer these questions: namely analyzing the evolution of the covariances and the observability of the system, that are related to its transition and observation Jacobians. This consideration ultimately boils down to an observability analysis of a (locally!) linear system. The unusual thing in this case here is that the system is non stationary. We will not address these aspects in this course.

## 2 Exercise: Beacon Localization with Kalman

We have a unicycle robot that we control in translational and rotational displacements $\Delta\rho$ and $\Delta\theta$ (see Appendix A). The robot moves on a plane, and the world is populated with two transmitting beacons. The robot can sense the beacon range and the beacon identities, and knows the location of the beacons a priori.

We want to localize the robot with an EKF. To solve the problem we need to:

- identify the state
- identify the controls
- identify the transition function
- identify the observations
- identify the observation function
- sketch a kalman filter that addresses the problem.

### 2.1 Domains

- State: $\mathbf{x} = (x\ y\ \theta)^T$, is an $SE(2)$ object, and we parametrize with a 3D vector. Normalization on angles will be required upon wraparounds.
- Control: $\mathbf{u} = (\Delta\rho\ \Delta\theta) \in \Re \times SO(2)$, parameterized by a 2D vector.
- Observations: we have 2 beacons at known locations $\mathbf{p}_1$ and $\mathbf{p}_2$, and we see them all the times. The observation vector $\mathbf{z} = (z_1\ z_2)^T \in \Re^{2+}$, stacks the sensed distances between the robot and the two landmarks.

## 2.2 Transition

- Transition Function: We use the transition in Eq 53, truncated at the first order:

$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \tag{2}$$

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} + \Delta\rho_{t-1}\cos\theta_{t-1} \\ y_{t-1} + \Delta\rho_{t-1}\sin\theta_{t-1} \\ \theta_{t-1} + \Delta\theta_{t-1} \end{pmatrix} \tag{3}$$

- Transition Jacobians:

$$\mathbf{A} = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \begin{pmatrix} 1 & 0 & -\Delta\rho\sin\theta \\ 0 & 1 & \Delta\rho\cos\theta \\ 0 & 0 & 1 \end{pmatrix} \tag{4}$$

$$\mathbf{B} = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \tag{5}$$

## 2.3 Observation

- Observation Function: we sense the range of the two beacons. If the robot is at position $\mathbf{x}$, we can predict the range at which it will sense a beacon in $\mathbf{p}_i$ as

$$\mathrm{h}_i(\mathbf{x}) = |\mathbf{t} - \mathbf{p}_i| \tag{6}$$

$$\tag{7}$$

  Here $\mathbf{t}$ represents the translational part of the robot position $\mathbf{x}$.

- Observation Jacobian: For one single beacon the Jacobian is the following a 1x3 matrix:

$$\mathbf{C}_i = \frac{\partial \mathrm{h}_i(\mathbf{x})}{\partial \mathbf{x}} = -\frac{1}{|\mathbf{t} - \mathbf{p}_i|} \begin{pmatrix} x - x_i & y - y_i & 0 \end{pmatrix} \tag{8}$$

- Complete observation model and Jacobians: is obtained by stacking observation models for each beacon. Same considerations as the one made for the observation functions apply to the complete Jacobian:

$$\mathbf{h}(\mathbf{x}) = \begin{pmatrix} \mathrm{h}_1(\mathbf{x}) \\ \mathrm{h}_2(\mathbf{x}) \end{pmatrix} \tag{9}$$

$$\mathbf{C} = \begin{pmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{pmatrix} \tag{10}$$

## 2.4 Filter

We use an EKF. These are the steps:

**Predict** A new control $\mathbf{u}_{t-1}$ comes in, and we need to update our current belief by integrating the control

- $\mu_{t|t-1} = \mathbf{f}(\mu_{t-1|t-1}, \mathbf{u}_{t-1})$: predict the mean, based on Eq. 2.
- $\mathbf{A}_t = \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$: linearize the transition w.r.t. the state, based on Eq. 4.
- $\mathbf{B}_t = \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}$: linearize the transition w.r.t. the control, based on Eq. 5.
- $\mathbf{\Sigma}_{t|t-1} = \mathbf{A}_t \mathbf{\Sigma}_{t-1|t-1} \mathbf{A}_t^T + \mathbf{B}_t \mathbf{\Sigma}_u \mathbf{B}_t^T$: predict the covariance (affine transform on Gaussians, from the linearized model).

**Update** A new observation $\mathbf{z}_t$ comes in, and we need to update our predicted belief by integrating the measurement

- $\mu_z = \mathbf{h}(\mu_{t|t-1})$: predict the mean of the observation, based on Eq. 9
- $\mathbf{C}_t = \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$: linearize the observation function, based on Eq. 10.
- $\mathbf{K}_t = \mathbf{\Sigma}_{t|t-1} C_t^T (\mathbf{\Sigma}_z + C_t \mathbf{\Sigma}_{t,t-1} C_t^T)^{-1}$, Kalman gain (see slides).
- $\mu_{t|t} = \mu_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mu_z)$, mean update (slides).
- $\mathbf{\Sigma}_{t|t} = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t)\mathbf{\Sigma}_{t|t-1}$ covariance update (slides).

## 3 Exercise: Watch my robot

We have a unicycle robot that is controlled in translational and rotational displacements $\Delta\rho$ and $\Delta\theta$. The displacements (so the velocities) vary mildly, but we do not have access to them. The robot is observed by $N$ stations located at known position. Each station senses the "bearing of the robot"

We want to track the robot with an EKF (determine its position).

### 3.1 Domains

- State: the state now is the position of the robot *and* its displacements, since we don't observe them, but we can assume they are change mildly. The state $\mathbf{x} = (x \; y \; \theta \; \Delta\rho \; \Delta\theta) \in SE(2) \times \Re \times SO(2)$ is represented as a 5D vector.
- Controls: none
- Observations: we get a stack of bearing measurements $\mathbf{z} = (z_1 \; \cdots \; z_N) \in SO(2)^N$, that we represent with an N dimensional vector;

### 3.2 Transition

- Transition Function: We use the transition of the section, truncated at the first order:

$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}) \tag{11}$$

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \\ \Delta\rho_t \\ \Delta\theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} + \Delta\rho_{t-1}\cos\theta_{t-1} \\ y_{t-1} + \Delta\rho_{t-1}\sin\theta_{t-1} \\ \theta_{t-1} + \Delta\theta_{t-1} \\ \Delta\rho_{t-1} \\ \Delta\theta_{t-1} \end{pmatrix} \tag{12}$$

- Transition Jacobian ($\mathbf{B}$ is not needed)

$$\mathbf{A} = \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}} = \begin{pmatrix} 1 & 0 & -\Delta\rho\sin\theta & \cos\theta & 0 \\ 0 & 1 & \Delta\rho\cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{13}$$

$$\tag{14}$$

### 3.3 Observation

- Observation Function: the observation made by a station depends only on the translation of the robot, not on its orientation. If the robot at pose $\mathbf{x}$ is sensed by a station at position $\mathbf{p}_i$, we will measure the angle of the vector difference between the two:

$$\mathrm{h}_i(\mathbf{x}) = \mathrm{atan}\frac{y - y_i}{x - x_i} \tag{15}$$

$$\tag{16}$$

Here $x_i$ and $y_i$ are he coordinates of the station.

- Observation Jacobian: For one single beacon the Jacobian is the following a 1x5 matrix:

$$\mathbf{C}_i \;=\; \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} = \frac{1}{(x-x_i)^2+(y-y_i)^2}\begin{pmatrix} y_i - y & x - x_i & 0 & 0 & 0 \end{pmatrix} \tag{17}$$

- Complete observation model and Jacobians: is obtained by stacking observation models for each beacon. Same consideration apply to the Jacobian:

$$\mathbf{h}(\mathbf{x}) \;=\; \begin{pmatrix} \mathrm{h}_1(\mathbf{x}) \\ \vdots \\ \mathrm{h}_N(\mathbf{x}) \end{pmatrix} \tag{18}$$

$$\mathbf{C} \;=\; \begin{pmatrix} \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_N \end{pmatrix} \tag{19}$$

### 3.4 Filter

We use an EKF. These are the steps:

**Predict** A new control $\mathbf{u}_{t-1}$ comes in, and we need to update our current belief by integrating the control

- $\mu_{t|t-1} = \mathbf{f}(\mu_{t-1|t-1}, \mathbf{u}_{t-1})$: predict the mean, based on Eq. 11.
- $\mathbf{A}_t = \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$: linearize the transition w.r.t. the state, based on Eq. 13.
- $\mathbf{\Sigma}_{t|t-1} = \mathbf{A}_t \mathbf{\Sigma}_{t-1|t-1} \mathbf{A}_t^T + \Sigma_{\text{process}}$: predict the covariance (affine transform on Gaussians, from the linearized model). Here we made a little hack by adding a covariance $\Sigma_{\text{process}}$, that is a small constant diagonal matrix. These entries aim at "injecting" some noise in estimate since we are assumed the velocities to be stationary while they are not. This serves to prevent overconvergence of the filter.

**Update** A new observation $\mathbf{z}_t$ comes in, and we need to update our predicted belief by integrating the measurement

- $\mu_z = \mathbf{h}(\mu_{t|t-1})$: predict the mean of the observation, based on Eq. 18
- $\mathbf{C}_t = \frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}$: linearize the observation function, based on Eq. 19.
- $\mathbf{K}_t = \mathbf{\Sigma}_{t|t-1} C_t^T (\mathbf{\Sigma}_z + C_t \mathbf{\Sigma}_{t,t-1} C_t^T)^{-1}$, Kalman gain (see slides).
- $\mu_{t|t} = \mu_{t|t-1} - \mathbf{K}_t(\mathbf{z}_t - \mu_z)$, mean update (slides). Remember to normalize the difference between angles $\mathbf{z}_t - \mu_z$, otherwise your system will not work.
- $\mathbf{\Sigma}_{t|t} = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t)\mathbf{\Sigma}_{t|t-1}$ covariance update (slides).

## 4 Exercise: 2D GPS with global clock

We have a unicycle robot that is controlled in translational and rotational displacements $\Delta\rho$ and $\Delta\theta$, that we can access. The robot is equipped with an antenna capable of receiving the signal emitted by a set of beacons, that move on the plane. In the message, each beacon encodes:

- unique ID
- its position on the plane $\mathbf{p}_i$

- the time $t_i^s$ at which the beacon transmits the signal, on a global clock.

The robot might not sense all the beacons at all times.

We want to track the robot with an EKF (determine its position), under the assumption that it has access to the global clock. In this case, we can assume that, when the robot receives a message it immediately timestamps it to get the arrival time $t_i^a$. We remark that knowing the location of the beacon $\mathbf{p}_i$, the sending time $t_i^s$, and the robot location $\mathbf{x}$, we can compute the expected arrival time.

## 4.1 Domains

- State: is a pose on a plane $\mathbf{x} = (x\ y\ \theta)^T$, as described in Section 2.1.
- Controls: two displacemets a pose on a plane $\mathbf{u} = (\Delta\rho\ \Delta\theta)^T$, as described in Section 2.1.
- Observations: we observe a time (or a time difference). A single beacon observation $z_i \in \Re$ is a scalar. In case of multiple observations from the same location, we stack in a vector $\mathbf{z} = (z_1\ \cdots\ z_N)^T$.

## 4.2 Transition

The transition model and jacobian are the one described in Section 2.2.

## 4.3 Observation

- Observation Function: depends on what we choose as measurement to be. We can model the problem by using a predicted measurement that is the global time, or that is the time difference between arrival and sending. To proceed in the synthesis, imagine you know perfectly the state, and compute what time (or time difference our robot would measurement). Recall that the beacon position $\mathbf{p}_i$ is known and encoded in the message, and so the transmission time $\mathbf{t}_i^s$ is. If the robot is at location $\mathbf{x}$, it will receive the message at time:

$$t_i^r = \frac{1}{c}|\mathbf{t} - \mathbf{p}_i| + t_i^s. \tag{20}$$

  Here $\mathbf{t}$ is the translational part of the robot pose and $c$ is the speed of the light.

  Since $t_i^s$ is a constant in our problem, we can model the measurement as a time difference,

$$z_i = \mathrm{h}_i(\mathbf{x}) = t_i^r - t_i^s = \frac{1}{c}|\mathbf{t} - \mathbf{p}_i|. \tag{21}$$

  which leads to a more numerically stable computation. The whole observation function consists of a vector obtained by stacking the different $\mathrm{h}_i(\mathbf{x})$

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) = \begin{pmatrix} \mathrm{h}_1(\mathbf{x}) \\ \vdots \\ \mathrm{h}_N(\mathbf{x}) \end{pmatrix}. \tag{22}$$

  Since we do not know how many beacons we observe at a time, we need to assemble such a vector dynamically, that means that the size of our observation domain will vary in time. We need to assemble the kalman filter dynamically, by composing a measurement vector and a measurement function based on the actual measurements we receive in a time epoch.

- Observation Jacobian: We observe the similarity between our measurement function, and the one in Section 2.3. The difference is just the division by the speed of the light. For one single beacon the Jacobian is the following a 1x3 matrix:

$$\mathbf{C}_i = \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} = -\frac{1}{c|\mathbf{t} - \mathbf{p}_i|} \begin{pmatrix} x - x_i & y - y_i & 0 \end{pmatrix} \tag{23}$$

The total Jacobian is the stack of the measurement jacobians:

$$\mathbf{C} = \begin{pmatrix} \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_N \end{pmatrix} \tag{24}$$

### 4.4 Filter

We use an EKF. These are the steps:

**Predict** The predict step is completely equivalent to the one in Section 2.4.

**Update** As stated before, we do not observe all beacons simultaneously. The only aspect here is that the measurement vector $\mathbf{z}$, the measurement function $\mathbf{h}(\mathbf{x})$ and the measurment Jacobian $\mathbf{C}$ are assembled by stacking the available measurements. If at a certain interval we have the beacons 1, 3 and 7 we will have

$$\mathbf{z} = \begin{pmatrix} z_1 \\ z_3 \\ z_7 \end{pmatrix} \qquad \mathbf{h}(\mathbf{x}) = \begin{pmatrix} \mathrm{h}_1(\mathbf{x}) \\ \mathrm{h}_3(\mathbf{x}) \\ \mathrm{h}_7(\mathbf{x}) \end{pmatrix} \qquad \mathbf{C} = \begin{pmatrix} \mathbf{C}_1 \\ \mathbf{C}_3 \\ \mathbf{C}_7 \end{pmatrix} \tag{25}$$

## 5 Exercise: 2D GPS without global clock

We have a unicycle robot that is controlled in translational and rotational displacements $\Delta\rho$ and $\Delta\theta$, that we can measure. The robot is equipped with an antenna capable of receiving the signal emitted by a set of beacons, that move on the plane. In the message, each beacon encodes:

- unique ID
- its position on the plane $\mathbf{p}_i$
- the time $t_i^s$ at which the beacon transmits the signal, on a global clock. The global clock is the same for all the beacons.

The robot might not sense all beacons at all time.

We want to track the robot with an EKF (determine its position), under the assumption the robot clock is affected by a small slowly changing bias $\tau$, such that the time measured by the robot is:

$$t_{\text{robot}} = t_{\text{global}} + \tau. \tag{26}$$

### 5.1 Domains

- State: is a pose on a plane $(x\ y\ \theta)^T \in SE(2)$, as described in Section 2.1 *and* a time drift $\tau$. Our state can be thus represented by a vector $\mathbf{x} = (x\ y\ \theta\ \tau)^T$. Note that, since we do not know the skew and we need that to compute our predicted observation, we include it in the state thus adding it to the estimation process.
- Controls: see Section 2.1.
- Observations: same as Section 4.1.

### 5.2 Transition

The transition model is similar to the one in Section 2.2, with the exception that we have also the quasi stationary time skew $\tau$.

- Transition Function: We use the transition of Section 2.2, augmented with the stationary bias:

$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \tag{27}$$

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \\ \tau_t \end{pmatrix} = \begin{pmatrix} x_{t-1} + \Delta\rho_{t-1}\cos\theta_{t-1} \\ y_{t-1} + \Delta\rho_{t-1}\sin\theta_{t-1} \\ \theta_{t-1} + \Delta\theta_{t-1} \\ \tau_{t-1} \end{pmatrix} \tag{28}$$

- Transition Jacobians:

$$\mathbf{A} = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} = \begin{pmatrix} 1 & 0 & -\Delta\rho\sin\theta & 0 \\ 0 & 1 & \Delta\rho\cos\theta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{29}$$

$$\mathbf{B} = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{30}$$

## 5.3 Observation

- Observation Function: If the robot is at location $\mathbf{x}$, it will receive the message at time:

$$t_i^r = \frac{1}{c}|\mathbf{t} - \mathbf{p}_i| + t_i^s + \tau. \tag{31}$$

  Here $\mathbf{t}$ is the translational part of the robot pose and $c$ is the speed of the light. Note that the time at which it senses the message is now depending on the bias.

  The measurement can be modeled as a time difference:

$$z_i = \mathrm{h}_i(\mathbf{x}) = t_i^r - t_i^s = \frac{1}{c}|\mathbf{t} - \mathbf{p}_i| + \tau. \tag{32}$$

  The same consideration about stacking the active measurements made in the previous exercise also hold for this one.

- Observation Jacobian: The state now has four components. For one single beacon the Jacobian is the following a 1x4 matrix:

$$\mathbf{C}_i = \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{x_i - x}{c|\mathbf{t}-\mathbf{p}_i|} & \frac{y_i - y}{c|\mathbf{t}-\mathbf{p}_i|} & 0 & 1 \end{pmatrix} \tag{33}$$

  Note that the Jacobian is now four dimensional, and it is consistent. The 0 in the third position means we cannot observe the effects of the changes in orientation. At the same time, the 1 in fourth position tells us that we can observe the bias $\tau$. The total Jacobian is the stack of the measurement jacobians, as usual

$$\mathbf{C} = \begin{pmatrix} \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_N \end{pmatrix} \tag{34}$$

## 5.4 Filter

We use an EKF. See the sections before. In this context, as in almost all exercises presented here, the behavior of the filter depends on the quality of the initial estimate. We will learn in a while how to approach this problem later on in this course. We always modeled highly nonlinear systems, and this renders the Kalman Filter non optimal. Observing the system, we note that the time skew cannot be controlled, only estimated. This results in the fact that sooner or later a straightforward implementation of this filter will converge to a value of $\tau$, and never move away from it. To prevent this, we can add periodically after the update a small element to the covariance element $\sigma_{\tau,\tau}$, to inject some noise in the time skew.

## A (Constant) Velocity Motion Model

In this section we will model the kinematics of a mobile robot that moves at constant velocities within a time interval. The platform is constrained to move along its (local) $x$ axis, that is assumed to be oriented forward, while the $y$ axis is assumed to be oriented towards the left.

The position of the platform on the plane is characterized by a transform $\mathbf{X} = (\mathbf{R}\ \mathbf{t})$, where $\mathbf{R}$ is a rotation matrix of an angle $\theta$, along the $z$ axis. With $\mathbf{t} = (x\ y)^T$ we refer to the translation vector. We can compactly express the location of the vehicle on the plane with a 3D vector $\mathbf{x} = (x\ y\ \theta)^T$. During a time interval $\Delta T$ the robot moves with constant translational velocity $v$, and constant rotational velocity $\omega$.

The first question we want to ask is where will our robot be after a time $\Delta T$? We answer to this question in two steps: first we assume the platform starts at the origin, then we extend the result for arbitrary locations.

### A.1 Robot in the origin

Our robot is at the origin and oriented along the $x$ axis of the plane ($\mathbf{x} = \mathbf{0}$). Maintaining constant velocities during this interval, the platform moves on an arc. The length of the arc $\Delta\rho$ is obtained by integrating the constant translational velocity during the interval. Similarly, the orientation $\Delta\theta$ is obtained by integrating the rotational velocity during $\Delta T$, as follows:

$$\Delta\rho = v\Delta T \tag{35}$$

$$\Delta\theta = \omega\Delta T \tag{36}$$

The robot is constrained to move along an arc of a circle of radius $R$, and its orientation should be tangent to the circle, thus we can write the following relation:

$$R\Delta\theta = \Delta\rho \tag{37}$$

From these three equations we recover the radius $R$ as

$$R = \frac{\Delta\rho}{\Delta\theta}. \tag{38}$$

This is consistent with the intuition that if the angular velocity is zero, the radius of the circle goes at infinity. We want now to determine the new position of the robot $\boldsymbol{\Delta}\mathbf{x} = (\Delta x\ \Delta y\ \Delta\theta)^T$ under these geometric constraints. We see that:

$$\Delta x = R\sin\Delta\theta \tag{39}$$

$$\Delta y = R(1-\cos\Delta\theta). \tag{40}$$

Note that $\Delta\theta$ can be calculated directly from Eq. 36. Yet, $R$ can go to infinity, and if not dealt appropriately, a direct implementation of the above equations might lead to numerical issues. Thus, we rework the above equations expanding $R$ as in Eq. 38:

$$\Delta x = \Delta\rho\frac{\sin\Delta\theta}{\Delta\theta} \tag{41}$$

$$\Delta y = \Delta\rho\frac{(1-\cos\Delta\theta)}{\Delta\theta}. \tag{42}$$

We notice that the functions $\frac{\sin\Delta\theta}{\Delta\theta}$ and $\frac{(1-\cos\Delta\theta)}{\Delta\theta}$ admit a finite limit for $\Delta\theta \to 0$. In general $\Delta T$ is small, thus also $\Delta\theta$ will be small. Accordingly, we can replace these two factors by their Tailor expansion around $\Delta\theta = 0$, which gives us the following

$$\frac{\sin\Delta\theta}{\Delta\theta} \simeq S(\Delta\theta) = 1 - \frac{1}{6}\Delta\theta^2 + \frac{1}{120}\Delta\theta^4 \tag{43}$$

$$\frac{1-\cos\Delta\theta}{\Delta\theta} \simeq C(\Delta\theta) = \frac{1}{2}\Delta\theta - \frac{1}{24}\Delta\theta^3 + \frac{1}{720}\Delta\theta^5. \tag{44}$$

Here $S(\Delta\theta)$ and $C(\Delta\theta)$ are two polynomials in $\Delta\theta$ containing the two Taylor expansions. Substituting them in Equations 41 and 42, we get the following:

$$\begin{aligned} \Delta x &= \Delta\rho S(\Delta\theta) && (45) \\ \Delta y &= \Delta\rho C(\Delta\theta) && (46) \end{aligned}$$

We will compactly write the function that maps a pair of translational and rotational displacements $\Delta\rho$ and $\Delta\theta$ as

$$\begin{aligned} \mathbf{\Delta x} &= \mathrm{m2t}(\Delta\rho, \Delta\theta) && (47) \\ \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{pmatrix} &= \begin{pmatrix} \Delta\rho S(\Delta\theta) \\ \Delta\rho C(\Delta\theta) \\ \Delta\theta \end{pmatrix} && (48) \end{aligned}$$

By increasing the, order of the expansion these approximations can be very accurate. Common implementations just stop the expansion at the first order, leading to

$$\begin{aligned} \Delta x &\simeq \Delta\rho && (49) \\ \Delta y &\simeq 0 && (50) \end{aligned}$$

For simplicity in the remainder of this document we will stick to this formulation, but when more accuracy is needed or the $\Delta\theta$ becomes large, using the higher order terms of $C$ and $S$ might prevent serous headaches.

## A.2 Robot not in the origin

If the robot starts at a generic location $\mathbf{X} = (\mathbf{R}\ \mathbf{t})$, we can use the results of the previous section, to obtain the next robot position $\mathbf{X}' = (\mathbf{R}'\ \mathbf{t}')$. On the plane this can be easily done by computing the motion of the robot as if it was starting from the origin by using Equation 48, and applying this motion from the location where the robot is. Let $\mathbf{\Delta x} = m2t(\Delta\rho, \Delta\theta)$, and $\mathbf{\Delta X} = (\mathbf{\Delta R}\ \mathbf{\Delta t})$ its corresponding transformation matrix, following:

$$\begin{aligned} \mathbf{R}' &= \mathbf{R} \cdot \mathbf{\Delta R} && (51) \\ \mathbf{t}' &= \mathbf{t} + \mathbf{R}\mathbf{\Delta t}. && (52) \end{aligned}$$

If we stop at the first order the expansion we can compactly express the new position vector as

$$\begin{aligned} x' &= x + \Delta\rho\cos\theta && (53) \\ y' &= y + \Delta\rho\sin\theta && (54) \\ \theta' &= \theta + \Delta\theta. && (55) \end{aligned}$$