Massachusetts Institute of Technology
Department of Mechanical Engineering

2.160 Identification, Estimation, and Learning
Fall 2022

Context-Oriented Project No. 2
Simultaneous Localization and Measurement

Out: October 17, 2022 Due: November 2, 2022
Study Group Discussions: October 27/28, 2022

In this context-oriented project, you will simultaneously estimate the pose (position and orientation) of a wheeled robot in a room, and the position and orientation of the 10 walls in the room. A map of the room is shown in Figure 2. First, you will estimate the pose trajectory of the robot based solely on odometry (wheel rotation measurement) data motor encoders. Next, you will integrate LIDAR data to leverage knowledge of the structure of the room to improve the pose estimate from that of odometry alone. This process simultaneously allows for improvement of the robot's model of the room. The Extended Kalman Filter will be the filter of choice for this analysis.

<u>How to read this COP</u>

Bolded text refers to deliverables that you are expected to perform. Completion of these tasks corresponds to points awarded.

Regular text refers to explanatory language that will help you better understand the problem or deliverables. This text may include tasks you need to perform for the sake of completing the deliverables, but these "support tasks" do not have points awarded.

Italicized text refers to hint text, which gives helpful bonus information --- you should be able to complete the COP with full understanding without ever reading these, but reading these may help you save time.

Section 1: Trajectory Prediction using Odometry

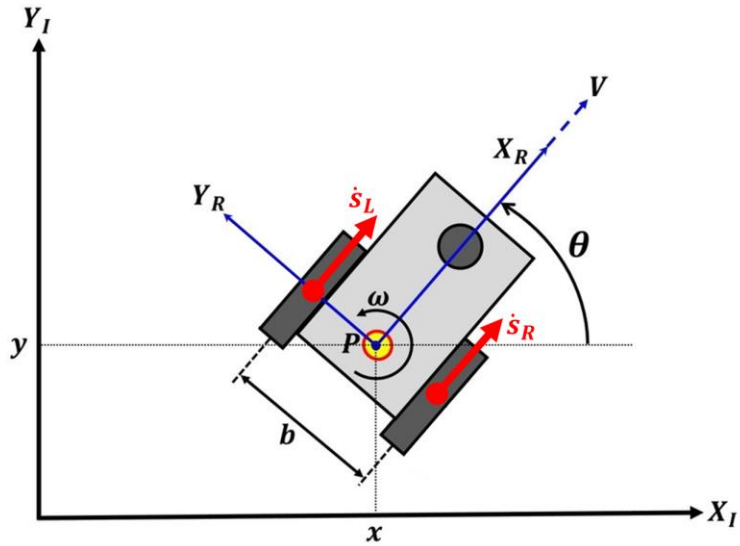

Figure 1 Diagram illustrating main odometry-related quantities

Refer to the SLAM Lecture Slides for a treatment of the kinematic relationship modeling motion of a two-wheeled robot over a time-step. You will use this model, along with the wheel path-distance data in `odom.csv`, to produce a prediction of the pose trajectory of the robot, which propagates covariance.

a) In order to use a Kalman Filter-like algorithm to propagate belief, it is important to have a Gaussian initial belief; that is, a belief based on an expected value (state estimate) and on belief covariance.

The central point of the robot, P in Figure 1, has a uniform probability of starting anywhere in a square starting region, marked red in the map in Figure 2. The initial orientation of the robot is independent of its starting position, and has a uniform probability of being between $-5\pi/9$ and $-4\pi/9$ radian (centered about pointing in the negative y direction).

Find the mean and covariance of the specified multivariate uniform initial belief. A multivariate Gaussian distribution with the same mean and covariance will form the initial belief used in the Extended Kalman Filter.

BY HAND

Hint: When random variables are independent, their joint distribution can be factored as a product of single-variable distributions.

Reference: Given $x\sim$ Unif(a, b), $\mathbb{E}[x] = \frac{1}{2}(a + b)$, and Var$[x] = \frac{1}{12}(b - a)^2$.

b) Download and examine `data\odom.csv`. Its contents are described in detail in `README.txt`. This file contains the accumulated lengths of the paths traced by the right and left wheels of the robot, s_R and s_L respectively. The wheel displacements Δs_R and Δs_L for each time step are calculated by subtracting s_R and s_L between time points.

Load `data\odom.csv` into MATLAB using the code provided in `example_data_loader.m`. You'll use `odom_struct` directly in Section 1.

Use these measurements (and the nonlinear discrete driving model) to generate a prediction algorithm to predict the pose trajectory of the robot, as well as propagate the estimation error covariance. For this prediction model, use an Extended Kalman Filter with no measurement updates implemented.

BY HAND

Implement your iterative prediction algorithm in the function `part_b_kalman_predict.m`, and use the code in the script `example_1.m` to properly run it using the loaded odometry data in `data_struct`.

Hint: If you code your Kalman prediction function to take in state vectors of arbitrary size, where the first three elements are the robot pose, then you will be able to reuse this function without modification in Section 2, Part k.

Plot both the estimated pose trajectory and covariance over time. It may be useful to examine these quantities both over time and across space. You might find the helper function `plot_2d_covariance_matrix.m` useful for visualizing the linear (x, y) covariance in real space.

WITH MATLAB

<u>Nonlinear discrete driving model:</u>

Inputs: Wheel turns $u_t = \left[\Delta s_{R,t}, \Delta s_{L,t}\right]^T$

States: Robot pose $\boldsymbol{x}_t = [x_t, y_t, \theta_t]^T$

Dynamic equation: $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, u_t) + G(\boldsymbol{x}_t, u_t) w_t$

Process noise covariance: $Q_t = \mathbb{E}[w_t w_t^T] = \begin{bmatrix} k_R\left|\Delta s_{R,t}\right| & 0 \\ 0 & k_L\left|\Delta s_{L,t}\right| \end{bmatrix}$

$$f(\boldsymbol{x}_t, u_t) = \begin{pmatrix} x_t + v_t c_\theta \\ y_t + v_t s_\theta \\ \theta_t + \Delta\theta_t \end{pmatrix}, \quad G(\boldsymbol{x}_t, u_t) = \begin{bmatrix} \frac{1}{2} c_\theta - \frac{v_t}{b} s_\theta & \frac{1}{2} c_\theta + \frac{v_t}{b} s_\theta \\ \frac{1}{2} s_\theta + \frac{v_t}{b} c_\theta & \frac{1}{2} s_\theta - \frac{v_t}{b} c_\theta \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix}$$

Using the following shorthand:

$c_\theta = \cos\left(\theta_t + \frac{1}{2}\Delta\theta_t\right)$, $s_\theta = \sin\left(\theta_t + \frac{1}{2}\Delta\theta_t\right)$, $v_t = \frac{1}{2}(\Delta s_R + \Delta s_L)$, $\Delta\theta_t = \frac{1}{b}(\Delta s_R - s_L)$

c) **Discuss the evolution of the pose estimate covariance matrix over time, and how it relates to the robot's activity.**

Section 2: SLAM using LIDAR

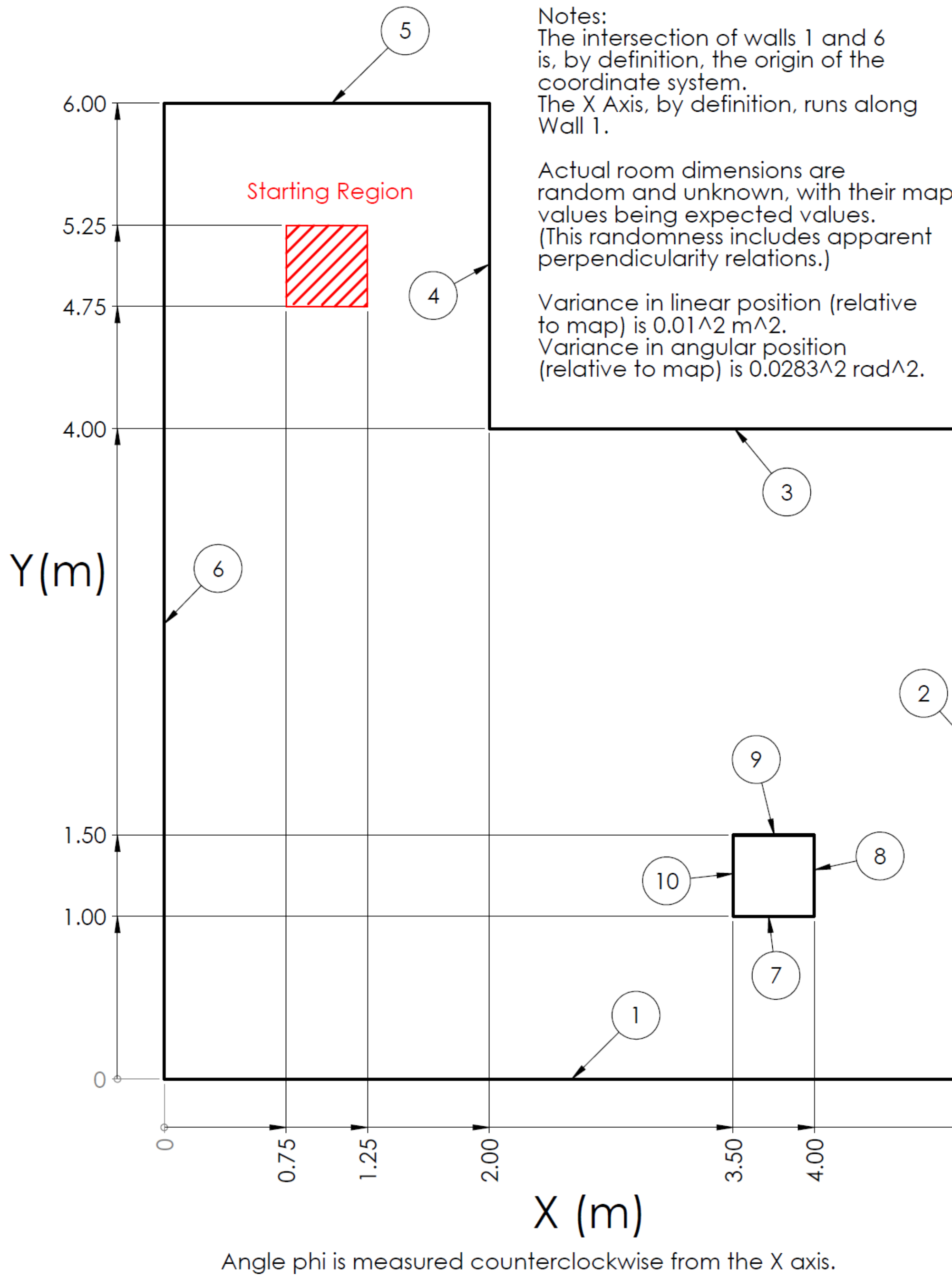

Figure 2 Map of the room traversed in this problem, with walls labeled

Now that you have an effective prediction model set up, the next step will be to integrate the structure of the room (i.e., the location of each of the 10 walls) and measurements by the LIDAR mounted on the robot (centered on point *P* in Figure 1, for simplicity), in order to correct the robot pose estimate, and to learn about the room.

Subsection 2.1: Setup and Theoretical Preliminaries

The first step in setting this up is to define an augmented state for estimation; as per the SLAM Lecture Slides, it consists of the robot pose, augmented with the distance parameter r_{map}^{j} and angle parameter α_{map}^{j} for the polar line representations of each wall $j \in \{1, \dots, 10\}$, as defined in the stationary reference frame of the map.

d) Based on the map in Figure 2 and the Notes therein, **generate an initial a priori estimate and initial covariance for the entire augmented state.** Assume that all elements of the initial a priori augmented state estimate are independent. BY HAND

Hint 1: You've already calculated the robot pose section of the initial a priori augmented state estimate and covariance. For the wall parameters section, use the dimensions in the map in Figure 2 to generate the state estimate. Refer to the Notes in the Figure for information on variance.

Hint 2: Consider how the map coordinate system is defined in terms of the walls. This should result in three specific wall parameters (representing two directions of translation and one direction of rotation) having zero variance relative to the map coordinate system.

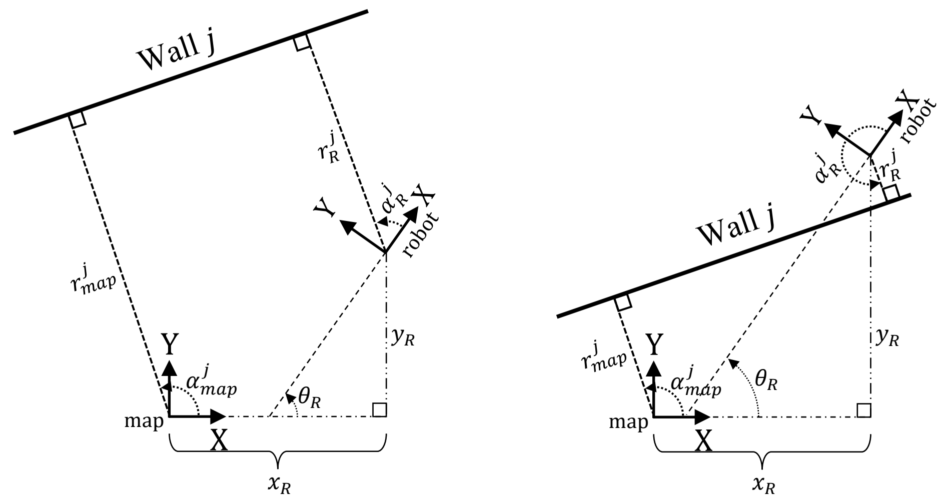

Figure 3. Relationship between map-reference-frame parameters and robot-reference-frame parameters

e) Given a general pose of the robot $(x_R, y_R, \theta_R)^T$, and given the map-reference-frame parameters r^j_{map} and α^j_{map} of a wall j, **calculate the robot-reference-frame parameters r^j_R and α^j_R of the same wall j.** This function, $\left(r^j_R, \alpha^j_R\right)^T = h_{wall}\left(x_R, y_R, \theta_R; r^j_{map}, \alpha^j_{map}\right)$, is the function that forms the basis for the measurement function h of the Extended Kalman Filter.
Now, it should be noted that the (r, α) line parameterization is properly defined with a nonnegative value of r. Make sure your formula always generates a nonnegative value of r; in order to do this, you will have to have some method of determining what side of the wall the robot is on, relative to the map origin (see Figure 3).

Also, calculate the partial Jacobian matrices for this wall H^j_R and $H^j_{map,j}$, defined as follows. These will be used later to construct the full measurement Jacobian H, which describes the measurement of all observed walls.

$$H^j_R := \begin{bmatrix} \frac{\partial r^j_R}{\partial x_R} & \frac{\partial r^j_R}{\partial y_R} & \frac{\partial r^j_R}{\partial \theta_R} \\ \frac{\partial \alpha^j_R}{\partial x_R} & \frac{\partial \alpha^j_R}{\partial y_R} & \frac{\partial \alpha^j_R}{\partial \theta_R} \end{bmatrix} \quad H^j_{map,j} := \begin{bmatrix} \frac{\partial r^j_R}{\partial r^j_{map}} & \frac{\partial r^j_R}{\partial \alpha^j_{map}} \\ \frac{\partial \alpha^j_R}{\partial r^j_{map}} & \frac{\partial \alpha^j_R}{\partial \alpha^j_{map}} \end{bmatrix}$$

It should be noted that, just like the equation for $h^j\left(x_R, y_R, \theta_R; r^j_{map}, \alpha^j_{map}\right)$, the equations for H^j_R and $H^j_{map,j}$ will differ depending on whether the map origin and the robot are on the same side of the wall.

BY HAND

Implement these formulas in the function
`part_e_observation_function_1_wall.m`.

WITH MATLAB

f) For the Extended Kalman Filter update step of SLAM, the measurement vector y_t is the vector of the robot-reference-frame parameters r_R^j, α_R^j of all the walls j that the LIDAR can see at time t. (These r_R^j, α_R^j are, in turn, estimated by fitting lines to the partitioned LIDAR data.)

For this question, assume that, at a given time t, three different walls, numbered i, j, k, are visible to the LIDAR. Assume that, for each wall, you have access to all the expressions from Part e, namely: the robot-frame parameters themselves and the two partial Jacobians. **Using these formulas from Part e, assemble the measurement function $\boldsymbol{h_t}$ such that $\boldsymbol{y_t} = \boldsymbol{h_t}(\boldsymbol{x_t})$**, where x_t is the full augmented state at time t. **Next, assemble the Jacobian matrix $\boldsymbol{H_t} \coloneqq \frac{\partial \boldsymbol{h_t}}{\partial \boldsymbol{x_t}}$.** (Note: This is the same H_t matrix used for updating the covariance and calculating Kalman gain in the Extended Kalman Filter.)

BY HAND

Implement this assembly for a general set of walls in the function `part_f_observation_builder.m`.

WITH MATLAB

Subsection 2.2: LIDAR Processing and Filtering

Download and examine `data\scan_dist.csv`, `data\scan_partition.csv`, and `data\scan_partition_labels.csv`. Their contents are described in detail in `README.txt`.

Load these files, as well as `data\odom.csv`, into MATLAB using the code provided in `example_data_loader.m`. You'll use `scan_struct` directly in Section 2 for dealing with scan processing, and `data_struct` for running the full Extended Kalman Filter at Part k.

Raw LIDAR data (i.e., the contents of `scan_dist.csv`) simply contain the measured distances of points along the LIDAR's angle sweep, without any metadata on what points correspond to what features in the environment. Normally when creating a SLAM algorithm, you would need to implement an algorithm to partition the raw LIDAR data into distinct features (or walls, in this example). A popular line-partitioning algorithm utilizes the *Hough transform* to extract features from an image. This is, however, outside of the scope of 2.160, so we've provided a partitioning for you to use, found in `scan_partition.csv`.

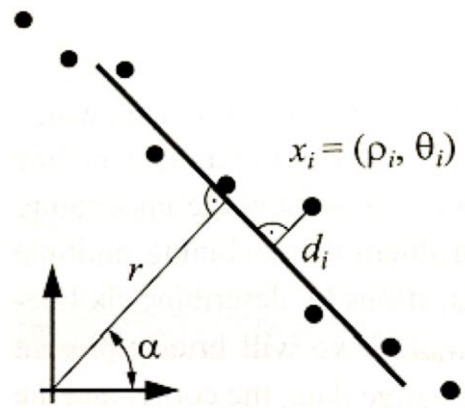

Figure 4. Illustration of the (r, α) parameterization of a line, with an illustration of the deviation d_i of a point i from the line. The dotted angles are right angles.

BY HAND, BUT IMPLEMENT IN MATLAB

g) **Define the error function to be minimized in `part_g_wall_param_fitter.m`, in order to achieve the following task:** Given a partition of N LIDAR data corresponding to a single wall, fit optimal robot-reference-frame line parameters r_R, α_R to that wall partition's data.

The MATLAB function performs sum-of-squares minimization, such that the following cost function is minimized:

$$J = \sum_i (e_i)^2 = \boldsymbol{e}^{\mathrm{T}}\boldsymbol{e}$$

The error function you need to provide to the MATLAB function is the vector $\boldsymbol{e}$ that participates in the inner-product vector definition of the sum-of-squares error J.

From your fit, this function (as written) also calculates the covariance matrix R_{wall} that corresponds to the vector of these two fit parameters.

Hint: Refer to Figure 4 and the SLAM Lecture Notes to derive and/or find this error function.

h) When processing the raw data of multiple walls to create usable parameter fits, the data of some walls ends up being too low-quality to be used effectively.

What might be a good criterion for throwing out ("pruning") raw point-cloud partition data *before* performing a parameter fit to the data?
This pre-fit pruning should serve to throw out point-clouds that cannot be properly fit using `part_g_wall_param_fitter.m`. This criterion should be based solely on the metadata attached to the raw point-cloud partition data (specifically, the number of points).

What might be a good criterion for pruning wall parameter fit data *after* performing a parameter fit to the data?
This post-fit pruning should serve to throw out wall parameters that are counter-productive to the Kalman update step in the SLAM; that is, we should throw out the walls for which the parameter fits are likely to be wrong. This criterion should be based solely the parameter fit data generated by `part_g_wall_param_fitter.m`: the parameter fit itself, and its fit covariance.
It should be noted that you won't be able to tune this criterion properly until you have completed the entire EKF-SLAM implementation, and have a way to evaluate its effectiveness (until you reach Parts l and m).

BY HAND, REFERENCING MATLAB CODE

Implement these pruning criteria in the function `part_h_full_scan_fitter.m`. (This function also utilizes `part_g_wall_param_fitter.m` to calculate all the wall parameters for a given scan.)

Complete the plotting equation in `part_h_fitted_scan_plotter.m` and use it plot to the complete partitioned LIDAR sweep at $t = 3.0\ sec$ from the data, plotting the partitioned sweep data and their fit lines.

WITH MATLAB

Discuss how the R_{wall} covariances differ for the different walls plotted from the $t = 3.0$ sec LIDAR sweep.

Discuss how different walls from the $t = 3.0\ sec$ LIDAR sweep may be included or excluded from the set used for the Kalman update, as a result of adjustment of your pruning criteria.

i) From the partitioning, we know which points belong to different walls within a single scan. That said, we don't necessarily know which walls in the room these walls actually correspond to.

For extra credit, implement in `part_i_fitted_scan_labeler.m` an algorithm that automatically assigns each wall partition in a scan to a "true" wall label, corresponding to the wall labels in the map in Figure 2. This algorithm is allowed to take as inputs the estimated full augmented state x_t, as well as the LIDAR data, and any post-processed output of the LIDAR data generated by functions you have created. You are NOT allowed to use `scan_partition_labels.csv`. *Hint: See the SLAM Lecture Slides on "Data Association".*

For regular credit, use the existing algorithm in `part_i_fitted_scan_labeler.m`, which uses `scan_partition_labels.csv` **to assign wall partitions to their corresponding true wall labels.**

EXPLAIN BY HAND, IMPLEMENT IN MATLAB

WITH MATLAB

j) **Complete the function `part_j_innovation_calculator.m` such that it performs the following task:**
At time t, given that we have a full augmented state estimate $\hat{x}_t$, we know which walls are observed by the LIDAR at time t, and we have access to each observed wall's LIDAR-estimated r_R, α_R, R_{wall}, construct or import the following things:

- The measurement vector, y_t
- The measurement covariance matrix, R_t
- The predicted measurement vector, $\hat{y}_t = h_t(\hat{x}_t)$ (from 2f)
- The measurement Jacobian (aka observation matrix) H_t (from 2f)
- The innovation vector $z_t = y_t - \hat{y}_t$
- The innovation covariance matrix $S_t = H_t P_{t|t-1} H_t^T + R_t$

This function is partly a generalization of the function calculated in 2f.

It should be noted that calculation of the innovation involves subtraction of angles, which must be done carefully; specifically, angle subtraction must only product results in the $[-\pi, \pi]$ range to be useful for the Kalman filter. Use the helper function `angle_subtract.m` to calculate your angle differences cleanly, instead of using the minus sign '-'.

k) **Implement the full Extended Kalman Filter update algorithm, referencing the other functions, in `part_k_kalman_update.m`.**

Alternate `part_b_kalman_predict.m` and `part_k_kalman_update.m` as needed to implement the complete Extended Kalman Filter for performing SLAM.

WITH MATLAB

Something to note is that the LIDAR scan measurement frequency is much lower than the odometry measurement frequency. Consequently, you will not be able to apply the EKF update from LIDAR every time step. This is not a problem, as the modular structure of discrete Kalman-like filters allows this. *(See the code in `example_2.m` for an example of how to run the EKF using the data from the data loader.)*

Subsection 2.3: Commentary

l) **Plot the SLAM-filtered estimated path of the robot, in the context of the final estimated walls of the room. How does this SLAM-filtered path compare to the path predicted by odometry alone?**

m) **Show how the covariance of the robot pose estimate changed over the course of the SLAM run. What events seemed to most greatly affect the robot pose covariance? Why do you think this was?**

n) **Extra credit: Examine how the covariance of the wall parameters changed over the course of the SLAM run. What events seemed to most greatly affect the wall parameter covariance? Why do you think this was?**

Bonus question (no credit, optional): **Do you have any recommendations regarding how this context-oriented project could be improved in future iterations?**

Appendix 1: Code Meta-Structure

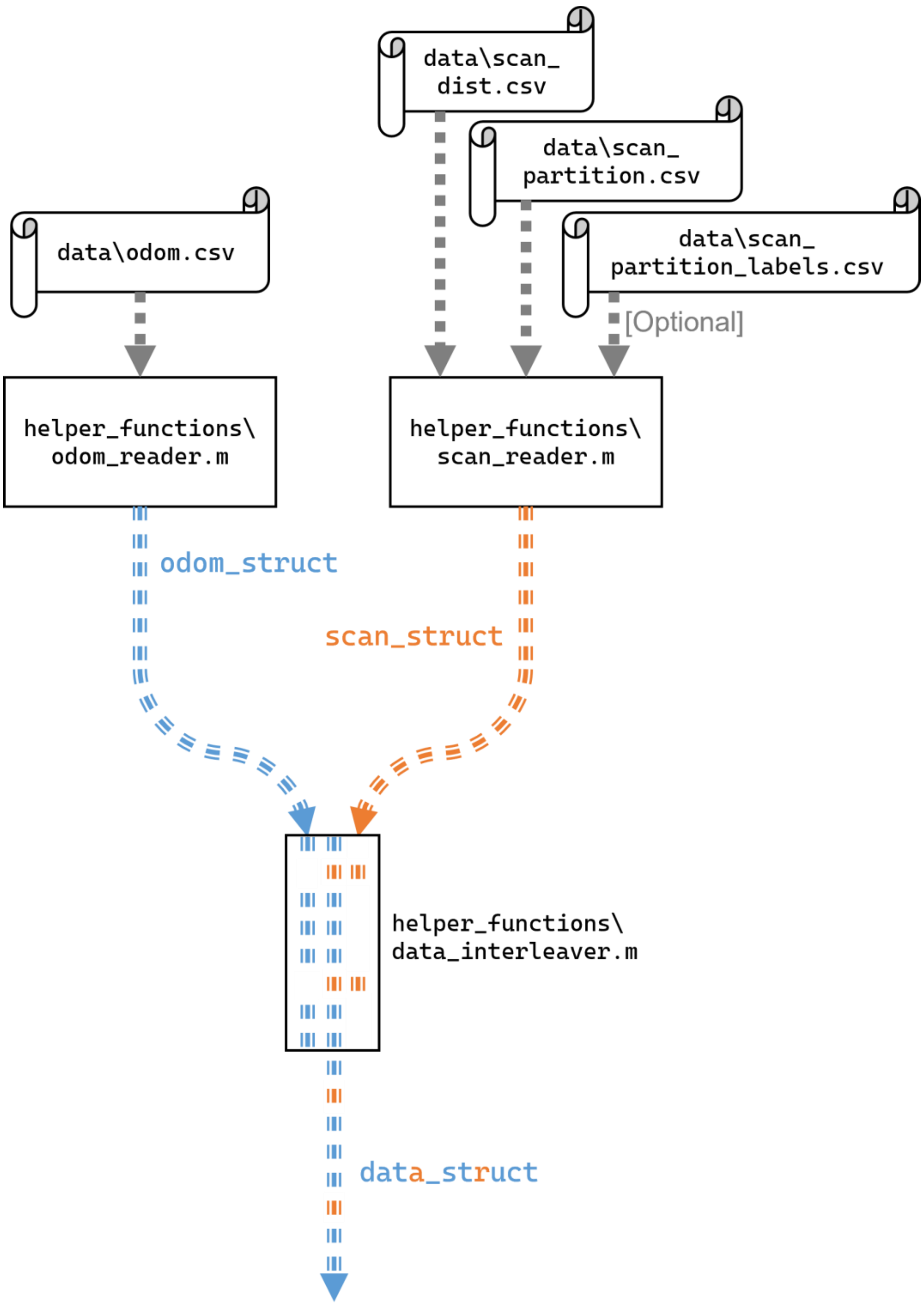

Figure 5. The code that generates the data structure **`data_struct`** from the source .csv files. You don't need to write any of this code – this is just for reference.

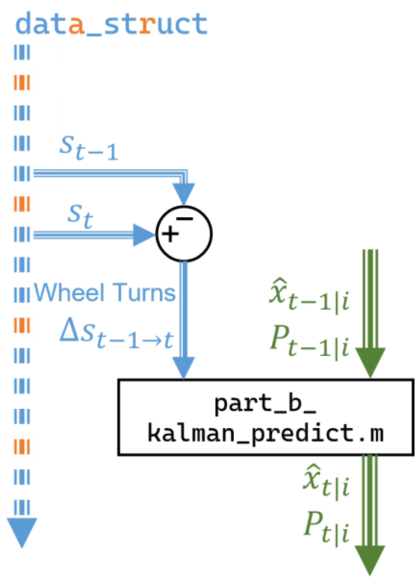

Figure 6. The Kalman prediction step that advances the state using odometry data

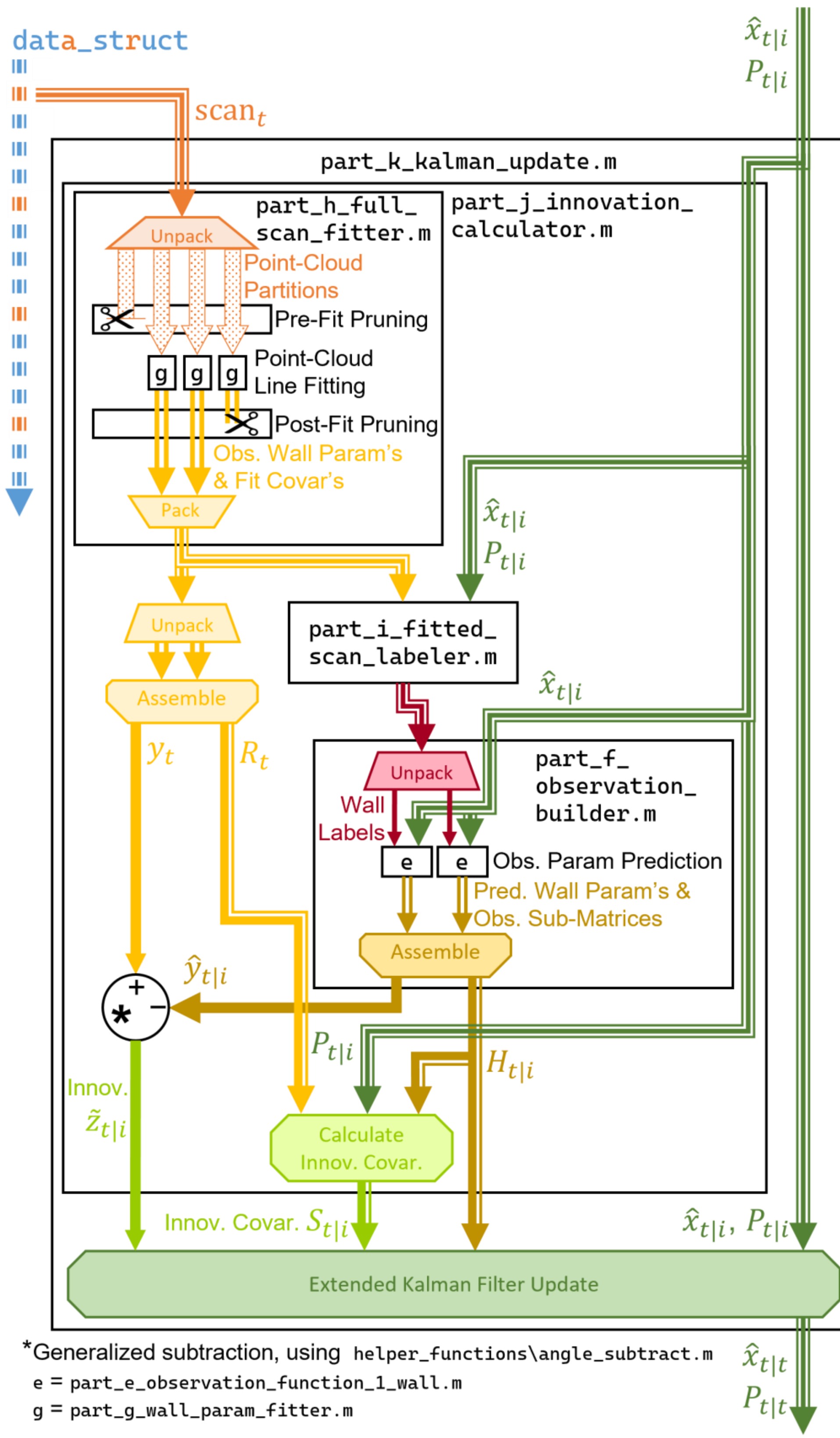

Figure 7. The Kalman update step that corrects the state using LIDAR scan data

Appendix 2: Function Inputs and Outputs

Function	Input Type	Output Type
`helper_functions\angle_subtract`	double (angular position), double (angular position)	double (angular difference)
`helper_functions\odom_reader`	string (.csv filename for file containing odometry data)	`odom_struct`
`helper_functions\scan_reader`	string (.csv … scan distance data), string (.csv … scan partition separation data), [optional] string (.csv … scan partition identity data)	`scan_struct`
`helper_functions\data_interleaver`	`odom_struct`, `scan_struct`	`data_struct`
`helper_functions\plot_2d_covariance_matrix`	2x2 double matrix (covariance matrix), [optional] 2x1 double vector (center point), [optional] double (Z-score, to scale plotted ellipse), [optional] double (line thickness for plotting), [optional] string OR 1x3 double vector (line color name or RGB triple (0-1))	none (action: plots 2D covariance matrix on existing figure)
`part_b_kalman_predict`	`state_estimate`, `wheel_turns`	`state_estimate`
`part_e_observation_function_1_wall`	3x1 double vector (robot pose), 2x1 double vector (map-frame wall parameters)	`predicted_wall_observation`
`part_f_observation_builder`	23x1 double vector (augmented state), N-dim integer vector (list of wall IDs)	`predicted_walls_observation`
`part_g_wall_param_fitter`	`data_struct.general.content{i}.partitions{j}` (assuming `data_struct.general.content{i}.type` = 'scan')	`wall_params`

Function	Input Type	Output Type
`part_h_full_scan_fitter`	`data_struct.general.content{i}` (assuming `data_struct.general.content{i}.type` = 'scan')	`fitted_scan`
`part_h_fitted_scan_plotter`	`fitted_scan` OR `fitted_labeled_scan`	none (action: plots the raw points belonging to the un-pruned fitted scans, as well as the fit lines themselves, with labels, if labels are present)
`part_i_fitted_scan_labeler`	`fitted_scan`, `state_estimate`	N-dim integer vector (list of wall IDs)
`part_j_innovation_calculator`	`data_struct.general.content{i}` (assuming `data_struct.general.content{i}.type` = 'scan'), `state_estimate`	`innovation_info`, `fitted_labeled_scan`
`part_k_kalman_update`	`state_estimate`, `data_struct.general.content{i}` (assuming `data_struct.general.content{i}.type` = 'scan')	`state_estimate`

Definitions of structs not directly defined by provided functions:

- `state_estimate`
  - `.state` 23x1 (or 3x1, for Section 1) double vector (augmented state estimate)
  - `.covariance` 23x23 (or 3x3, for Section 1) double matrix (augmented state estimate covariance)
- `wheel_turns`
  - `.right` double (right wheel turn between previous time step and this one)
  - `.left` double (left wheel turn between previous time step and this one)

Appendix 3: The Extended Kalman Filter Algorithm

Prediction Step: $\{\hat{\boldsymbol{x}}_{t-1|i}, \boldsymbol{P}_{t-1|i}\} \mapsto \{\hat{\boldsymbol{x}}_{t|i}, \boldsymbol{P}_{t|i}\}$ $(\boldsymbol{i} \leq \boldsymbol{t} - \mathbf{1})$

Given the following dynamic model:
$x_t = f_t(x_t) + G_t(x_t)w_t$
Where the process noise $w_t \sim \mathcal{N}(0, Q_t)$,
We have the following prediction algorithm:

Predicted state estimate: $\hat{x}_{t|i} = f_t\left(\hat{x}_{t-1|i}\right)$
Dynamics Jacobian: $F_{t|i} = \frac{\partial f_t}{\partial x}\Big|_{x\rightarrow\hat{x}_{t-1|i}}$
Predicted state estimate covariance: $P_{t|i} = F_{t|i}P_{t-1|i}F_{t|i}^{\mathrm{T}} + G_t\left(\hat{x}_{t-1|i}\right)Q_t G_t^{\mathrm{T}}\left(\hat{x}_{t-1|i}\right)$

Update Step: $\{\hat{\boldsymbol{x}}_{t|i}, \boldsymbol{P}_{t|i}\} \mapsto \{\hat{\boldsymbol{x}}_{t|t}, \boldsymbol{P}_{t|t}\}$ $(\boldsymbol{i} < \boldsymbol{t})$

Given the following measurement model:
$y_t = h_t(x_t) + v_t$
Where the measurement noise $v_t \sim \mathcal{N}(0, R_t)$,
We have the following update algorithm:

Innovation: $\tilde{z}_{t|i} = y_t - h_t\left(\hat{x}_{t|i}\right)$
Measurement Jacobian / Observation Matrix: $H_{t|i} = \frac{\partial h_t}{\partial x}\Big|_{x\rightarrow\hat{x}_{t|i}}$
Innovation Covariance: $S_{t|i} = H_{t|i}P_{t|i}H_{t|i}^{\mathrm{T}} + R_t$
Kalman Gain: $K_{t|i} = P_{t|i}H_{t|i}^{\mathrm{T}}S_{t|i}^{-1}$
Updated state estimate: $\hat{x}_{t|t} = \hat{x}_{t|i} + K_{t|i}\tilde{z}_{t|i}$
Updated state estimate covariance: $P_{t|t} = \left(I - K_{t|i}H_{t|i}\right)P_{t|i}$