CS469 HW0

Author: Conor Hayes
Repo clone URL: git@github.com:cwoodhayes/cs469-hw0.git
Dataset: ds1

Part A

1. Design a motion model to estimate the positional (2-D location and heading) changes of a wheeled mobile robot in response to commanded speed (translational and rotational) controls. Clearly define the inputs, outputs and parameters of this model, and report the math. Explain your reasoning, as needed. Is this model linear in its inputs? Why or why not?

A deterministic model of the motion of this car was derived using standard differential-drive robot kinematics, following the model described in *Probabilistic Robotics, Section 5.5.3*, following which random variables representing uncertainty were applied to the output.
This model approximates a trajectory by assuming that the robot follows a sequence of curved paths, each of which is an arc along a circle (or a straight line in the case of **ω**=0).

Motion model inputs:

- $\mathbf{x_{t-1}} = (x_{t-1}, y_{t-1}, \theta_{t-1})$ - cartesian coordinates in meters, and angular orientation in radians, for the car's position at the start of this dt
- **v** - forward velocity in m/s (scalar)
- **ω** - angular velocity in rads/s (scalar)
- **dt** - timestep (s)
- $\boldsymbol{\alpha} = (\boldsymbol{\alpha_1}, \boldsymbol{\alpha_1}, \boldsymbol{\alpha_1})$ - noise variables; 3 independent random variables sampled from a gaussian with variance **σ**

Motion model outputs:

- $\mathbf{x_t} = (x_t, y_t, \theta_t)$ - cartesian coordinates in meters, and angular orientation in radians, for the car's position at the end of this dt

Motion model parameters:

- **σ -** the standard deviation of a gaussian from which the motion noise **α** is sampled

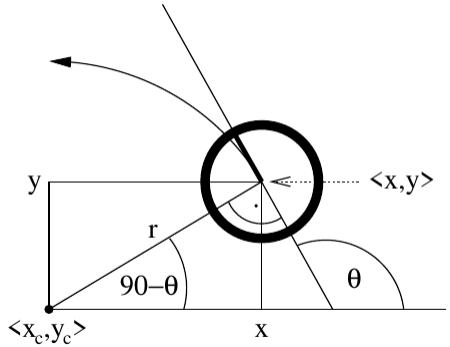

Fig 1.1 - Diagram showing how this model is derived. The robot follows the curved path angled up and to the left, which is an arc of a circle with center $\langle x_c, y_c \rangle$

and radius *r*. The equations written below follow the trajectory around this arc.

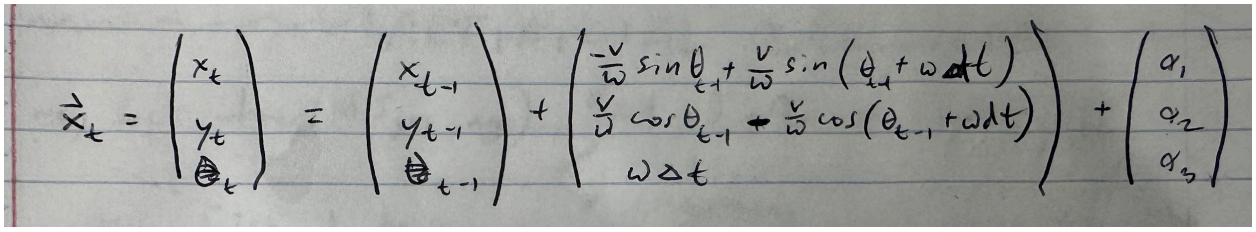

Fig 1.2 - equations of motion for the model

Note that in this model, the noise is applied to the deterministic output state resulting from the left 2 terms. Aside from the very first sample, this is equivalent to incorporating the noise at the inputs, and slightly simpler to implement/reason about. A more flexible representation of the noise might draw **α** from a 3D gaussian with a 3x3 covariance matrix, rather than 3 univariate distributions with variance **σ**. However, empirically, this simpler form works quite well on the sample data under test, and so was left undisturbed.

2. Implement your motion model in code, and observe the movements it generates for the following sequence of commands. Report the values of any parameters. (Below, v is translational speed, ω is rotational speed, and t is the duration of the commands.) Report the resulting plot.

$$[v=0.5\frac{m}{s},\ \omega=0\frac{rad}{s},\ t=1s]$$
$$[v=0\frac{m}{s},\ \omega=\frac{-1}{2\pi}\frac{rad}{s},\ t=1s]$$
$$[v=0.5\frac{m}{s},\ \omega=0\frac{rad}{s},\ t=1s]$$
$$[v=0\frac{m}{s},\ \omega=\frac{1}{2\pi}\frac{rad}{s},\ t=1s]$$
$$[v=0.5\frac{m}{s},\ \omega=0\frac{rad}{s},\ t=1s]$$

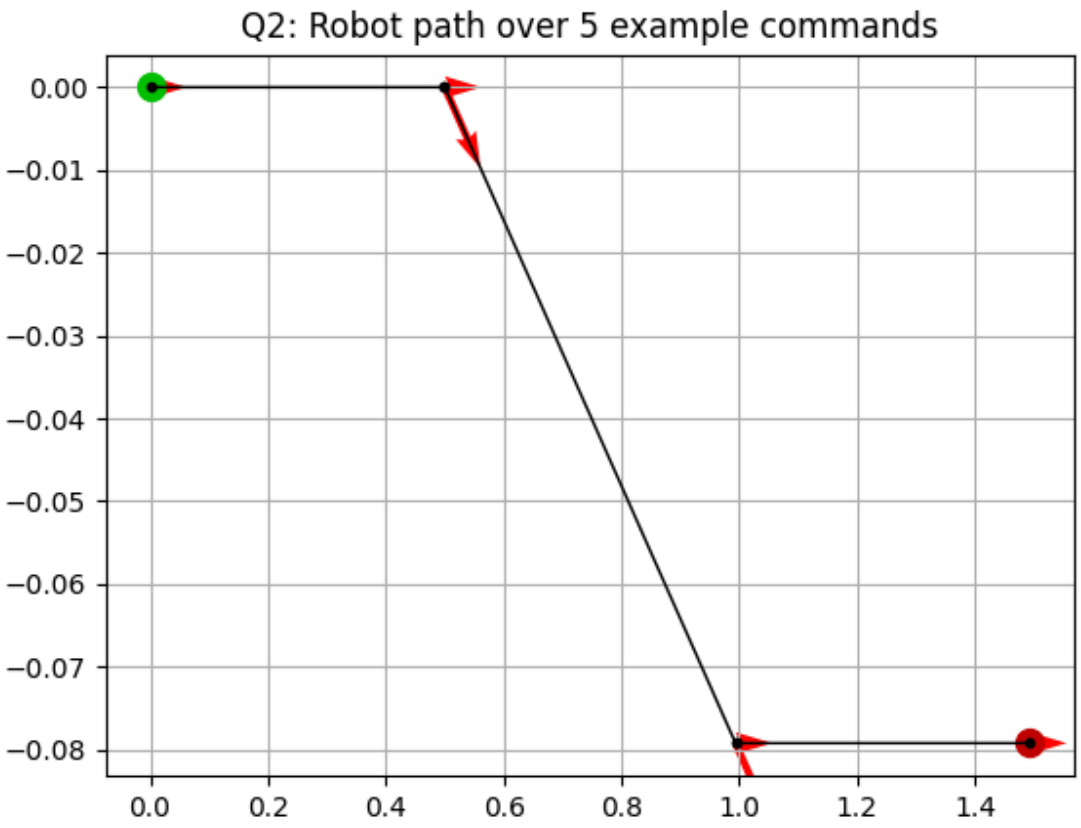

Fig 2.1 - Robot path following the commands described above, with parameter σ**=0**

To reach this result, a noiseless, deterministic version of the model (i.e. σ**=0**) was used.

3. Test your motion model on the robot dataset. Issue the controls commands (in _Controls.dat) to your model, and compare this dead-reckoned path (i.e. controls propagation only) to the ground truth path. Report and discuss the resulting plot.

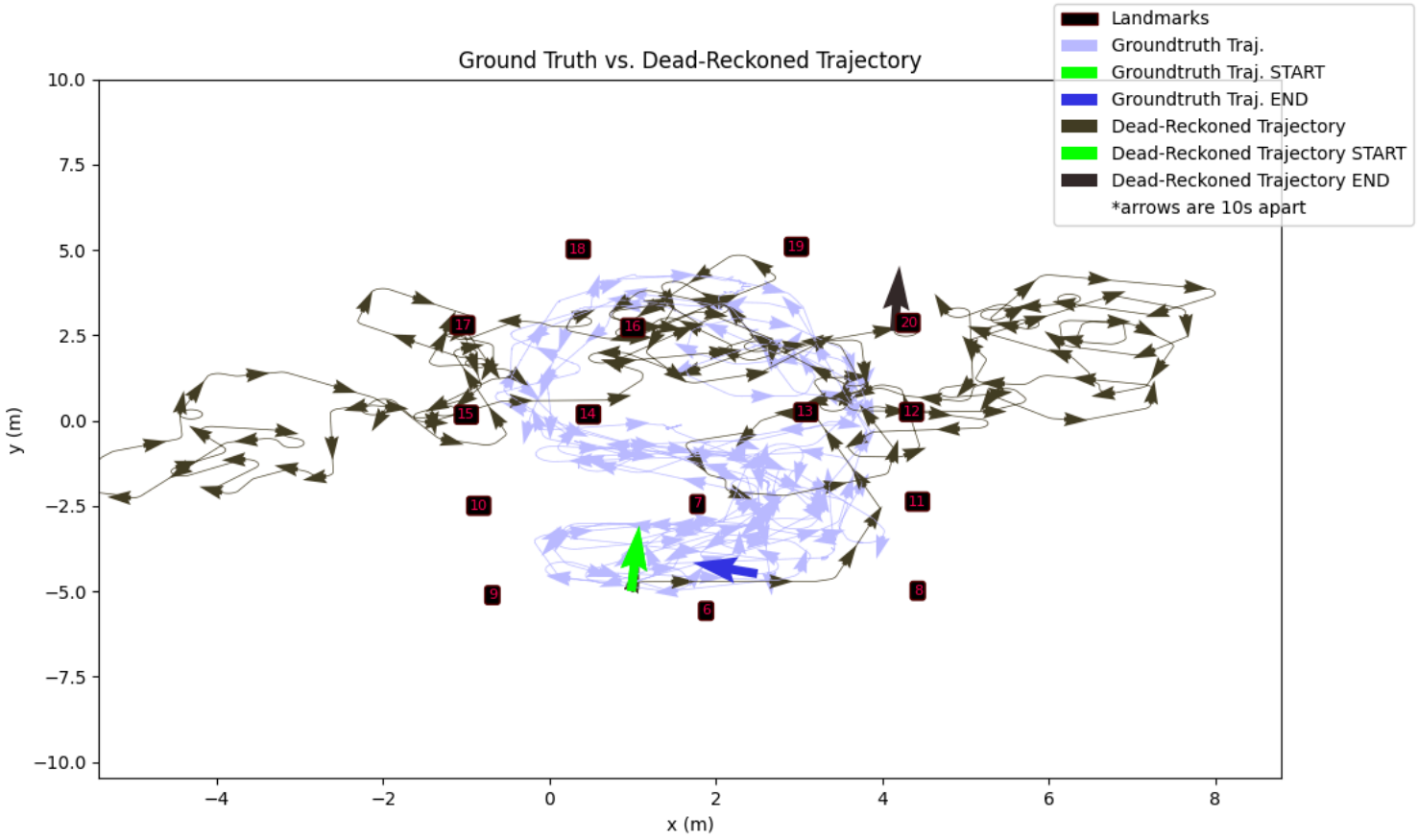

Fig 3.1 - A plot of the ground-truth trajectory (light blue) alongside the dead-reckoned trajectory from empirical measurements (dark brown). Landmarks are shown as black rounded rectangles & labeled with their subject number. The (same-colored) arrows on each path occur every 10 seconds, enabling visual comparison of the commanded vs. actual velocities as well.

As shown in the plot above, the deterministic “dead dead reckoning” trajectory differs significantly from the ground truth.
This is due to a variety of factors largely under the heading of “control uncertainty”, which we don’t have access to due to the limited nature of the dataset, but likely includes factors such as motor & gear slippage, encoder noise, and others. Over the course of each timestep, this uncertainty causes the ideal controlled trajectory to deviate slightly from reality, and these errors compound as time goes on to result in a completely different path.

4. Describe, concisely, the operation of your assigned filtering algorithm. Include any key insights, assumptions, requirements. Use equations in your explanation.

The particle filter is a computationally tractible approximation of an ideal Bayesian filter. Like other such filters, a “belief distribution” **bel(x_t)** describing the system state at time *t*, factoring in controls **u** and measurements **z**, is recursively updated over some timestep *dt*. This occurs over two rough steps:

1. *Control step:* An initial estimate of the new state **bel(x_t)** is reached using a state transition probability function **p(x_t | u_t , x_{t-1})** and the prior belief **bel(x_{t-1}).** In the case of the robot localization problem described in this assignment, this state transition function is derived from the system dynamics.
2. *Measurement step:* **bel(x_t)** is adjusted by factoring in measurements received during this timestep, using a measurement probability distribution **p(z_t | x_t)**

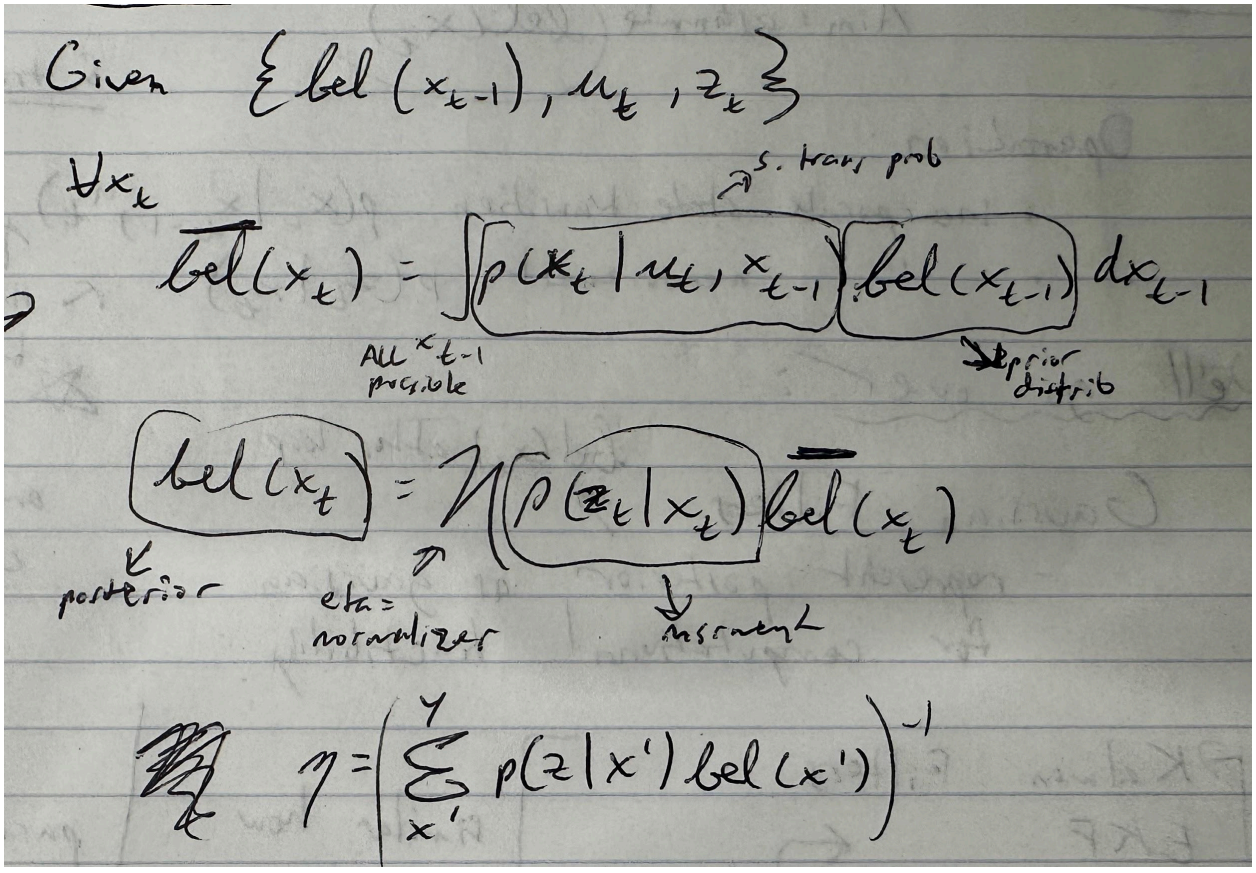

Fig 4.1 - Mathematical description of an ideal Bayes Filter.

Unlike many other Bayesian Filters (i.e. Kalman Filter, UKF, EKF, IF, etc), the particle filter represents its belief distribution not as a parametrized ideal distribution (i.e. a Gaussian), but as a set of **M** particles $\mathbf{X_t}$ sampled from that unknown belief distribution.

This takes advantage of the fact that although it may be impossible or computationally infeasible to represent an arbitrary distribution parametrically, we can often sample from such a distribution easily. In the case of the problem described here (Monte Carlo Localization), we can model a proposal distribution $\underline{\mathbf{bel}}(\mathbf{x_t})$ using a deterministic robot dynamics model to which noise (sampled from Gaussian, Triangle, Uniform, or other efficiently-sampleable distributions) is added.

After we sample from $\underline{\mathbf{bel}}(\mathbf{x_t})$ **M** times (giving us our **M** concrete state estimates AKA particles $\underline{\mathbf{X_t}}$), we can then update the distribution represented by these particles to $\mathbf{bel}(\mathbf{x_t})$ using "importance resampling", in which we evaluate the scalar value $\mathbf{p}(\mathbf{z_t} \mid \mathbf{x_{t,m}})$ for each particle $\mathbf{x_{t,m}}$, then use this new probability to weight each particle in $\underline{\mathbf{X_t}}$, after which they are randomly re-sampled with replacement. This results in the particle distribution shifting towards the actual $\mathbf{bel}(\mathbf{x_t})$.

1: **Algorithm Particle_filter**$(\mathcal{X}_{t-1}, u_t, z_t)$**:**
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Fig 4.2 - Pseudocode description of a particle filter. Step 4 is the control step, in which robot dynamics are incorporated, and Steps 5-10 contribute the measurement update. Steps 8-10 represent the "importance sampling" step.

5. Design a measurement model for your filter. Report all maths and reasoning; include images for illustration when appropriate.

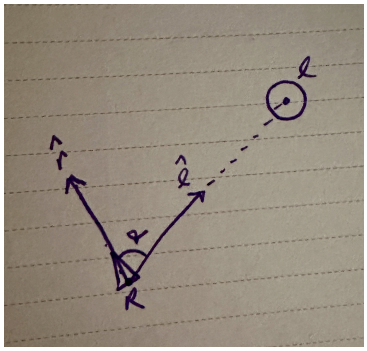

Fig 5.1 - A visual depiction of the measurement model. Here we have robot **R** facing in direction $\mathbf{r}_{hat}$ some distance away from a landmark **L**. The heading here is denoted as angle **α**, and the range is simply **| l |**.

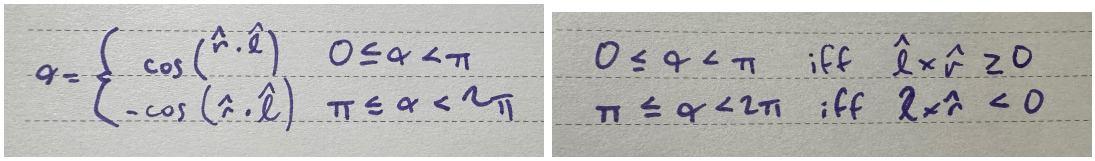

Fig 5.2 - The calculation of the heading angle, here denoted **α**

In this measurement model, we first generate a vector **l** from the robot to the landmark using **L - R**, and a unit robot heading vector $\mathbf{r}_{hat}$ via **<cosθ, sinθ>**, where **θ, R,** and **L** are known as part of the state & ground truth data, respectively.
Then, following the piecewise equations above, we find the heading **α** using the dot & cross product with **l** , and the range **| l |** using the 2-norm of **l**.

6. Test your measurement model on predicting the range and heading of the following landmarks when the robot is located at the associate positions. Report your results. (Remember, you have access to the ground truth landmark positions.)

Position (x,y,Θ)	Landmark (Subject #)	Predicted Range (m)	Predicted Heading (radians)
(2, 3, 0)	6	8.57	-1.5

(0, 3, 0)	13	4.13	-.729
(1, -2, 0)	17	5.22	1.97

PART B

7. Implement the full filter.

The full filter was implemented as described above, with the following implementation-specific choices made:

- Low-variance sampling was used when resampling particles from $\underline{\mathbf{X_t}}$ to calculate $\mathbf{X_t}$. Compared to selecting $\mathbf{X_t}$ using **M** random numbers, this uses a single random number to select across the whole set. In addition to being more efficient – *O(log***M***)* rather than *O(**M** * log***M**)* – it also maintains the diversity of the particle set (and thus, reduces the variance of the filter's output as a whole). The algorithm was implemented as described in *Probabilistic Robotics, Table 4.4*
- The proposal distribution $\underline{\mathbf{bel}}(\mathbf{x_t})$ was sampled from using the motion model following a Gaussian, as described in A.1 above. Gaussian was selected as a standard default choice, as it guarantees mathematically that $\underline{\mathbf{bel}}(\mathbf{x_t}) > 0 \ \forall \ \mathbf{x_t} \mid \mathbf{bel}(\mathbf{x_t}) > 0$, which is a necessary requirement for the resampling step to converge on $\mathbf{bel}(\mathbf{x_t}) > 0$ as **M** goes to infinity. However, other distributions such as triangles & uniforms are also mentioned as a possible choice in the literature, but were not explored here.
- All random number generation in the experiments below was initialized with a seed of 0, to make the experiments repeatable
- using a simple mean across all particles at each timestep. There are several alternatives to this approach (taking the median particle subject to some natural ordering, sampling randomly from $\mathbf{X_t}$, or taking an L2 norm instead of the L1 norm), but these were not explored.

Furthermore, the following decisions were made to accommodate the real-life dataset:

- In the ds1 dataset, many measurements were made against landmarks that did not have ground-truth locations associated with them (i.e. in ds1_Landmark_Groundtruth.dat). For the purposes of the filter tuning & experimentation, these measurements were discarded.
- The control & measurement frequencies in the dataset are different, especially after the removal of the measurements described above. Given this reality, several design questions must be answered in sequence:
    - Is the filter tick() run once per control, or once per measurement?
        - Decision: Run the filter once per control, and associate *(0, N]* measurements $\boldsymbol{z_{t,n}}$ with each control. Thus the measurement update step (lines 8-10 in Fig. 4.2) is actually run *n* times per control.

- What do we use for the input *dt* into the filter tick() function? The interval *(t-1, t)* between $\mathbf{u_{t-1}}$ and $\mathbf{u_t}$, or the interval *(t, t+1)* between $\mathbf{u_t}$ and $\mathbf{u_{t+1}}$?
    - Decision: Use *(t, t+1).* This is because each command is actually followed over the time period following that command being issued. The other option may lead to similar results if the control frequency is steady, but when the control frequency varies, it will drift from reality.
- How do we associate measurements with controls?
    - Each control $\mathbf{u_t}$ is associated with the *N* measurements $\boldsymbol{z_t}$ within *[t, t+1).* This is so that the assumption that measurements & controls occur simultaneously is closer to true (since our control is also assumed to apply over that same interval)

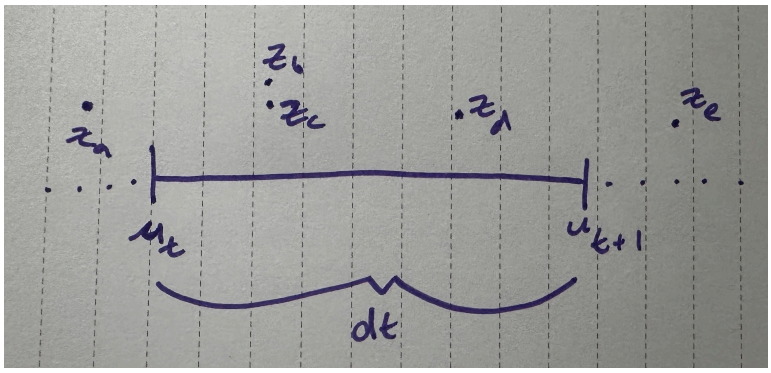

Fig 7.1 - The association between measurements and controls. In this diagram, the filter *tick()* function will receive as input at time **t** the following: ($\mathbf{u_t}$, $\mathbf{z_t}$, **dt**), where $\mathbf{z_t}$ **=** {$\mathbf{z_b}$, $\mathbf{z_c}$, $\mathbf{z_d}$}

8. [*] Compare the performance of your motion model (i.e., dead reckoning) and your full filter on the sequence of commands from step 2, and also on the robot dataset (as in step 3). Report the results, explain any differences (what performs well? what performs poorly? under what conditions? and why?). Ground your explanations in the algorithm maths.

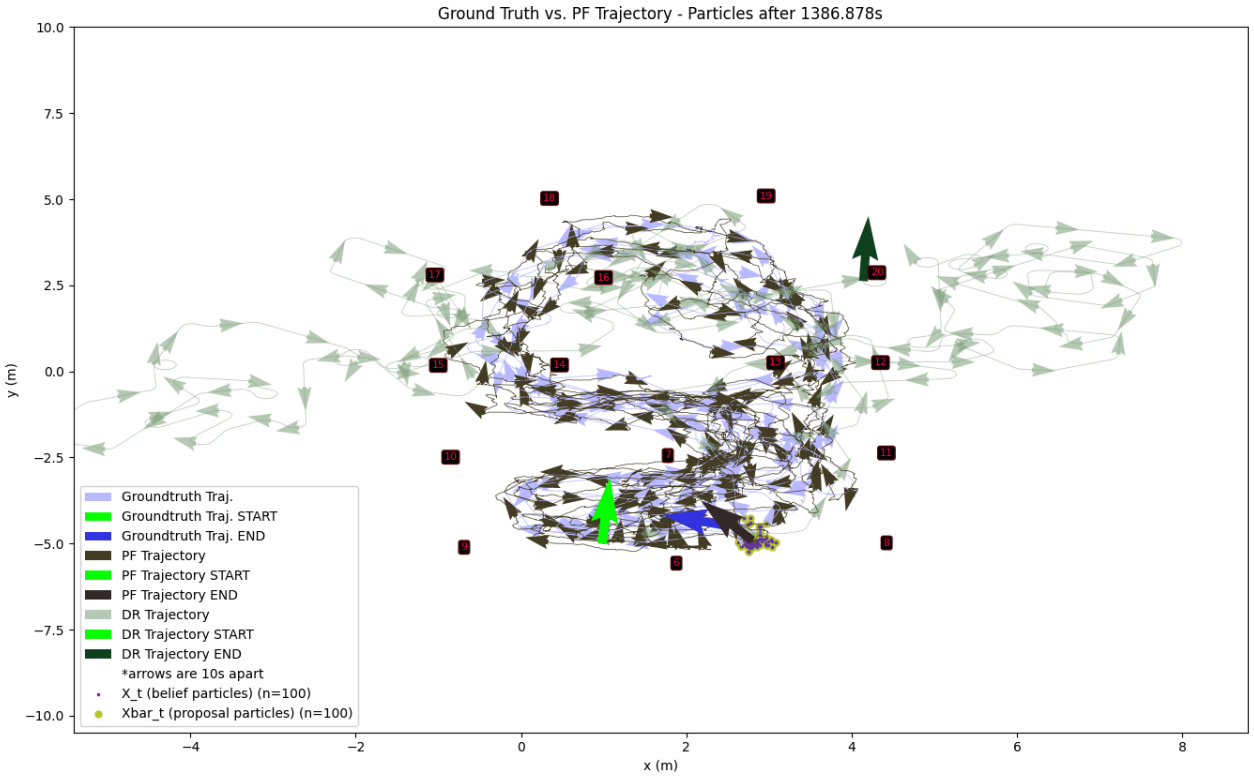

Fig 8.1 - The particle filter's operation given control & measurement data, compared against ground truth, with parameters set as shown in Fig 8.3 below.

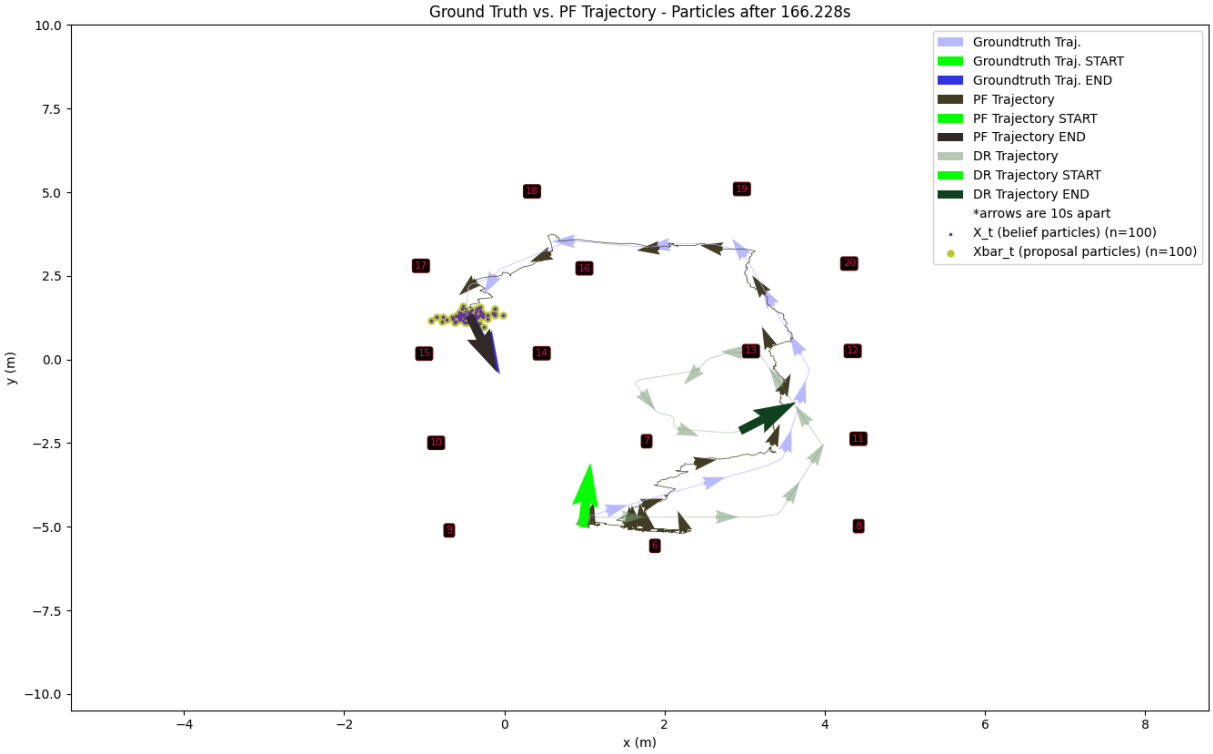

Fig 8.2 - The first 12% of the same trajectory shown in Fig 8.1. In this view, it is clearer how the predicted trajectory follows the ground truth (though there is some random noise introduced that makes the path quite jagged).

```
measure = MeasurementModel(
    ds.landmarks,
    cov_matrix=np.array(
        [
            [0.5, 0.2],
            [0.2, 0.5],
        ]
    )
    / 10,
)
u_noise = GaussianProposalSampler(
    stddev=0.05,
)
pf_config = ParticleFilter.Config(
    random_seed=0,
    n_particles=100,
)
```

Fig 8.3 - the parameters for noise, particles, and random seed used to arrive at the trajectory shown in Fig. 8.2

Overall, the algorithm performs quite well at following the path, with the correct tuning parameters. The method for arriving at these parameters is explored further in section **9** below. However, it does suffer from a jaggedness in the path, which potentially could be resolved by using a different method for arriving at predicted state $\mathbf{x}_t$ given output belief particle set $\mathbf{X}_t$ (other than the L1 norm used in this implementation). Another option could be to add an additional low pass filter step in series after the particle filter, to smooth its outputs.

9. [*] Examine the effect (trends? limits?) of uncertainty in the algorithm (i.e., changing the noise parameters). Examine its performance at different parts of the execution (e.g., when missing an expected measurement reading). Report plots and/or tables, and provide explanations.

In order to evaluate the performance of the model, a simple metric was defined to evaluate the distance between 2 trajectories (ground truth, and a trajectory generated by the filter parameters under evaluation).
The metric used in this analysis was a simple RMS error calculation, evaluated between the predicted trajectory and the ground truth trajectory (where the ground truth trajectory was linearly interpolated such that the timestamps matched up with the predicted trajectory's samples).

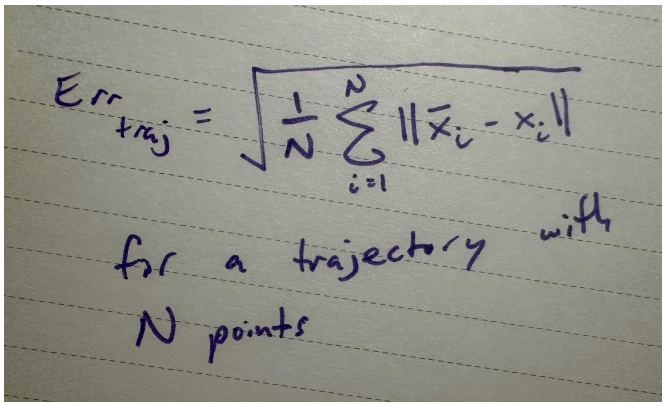

Fig 8.1 - RMS Trajectory Error calculation. In this image, the ground-truth trajectory represented by the points **x** has been linearly interpolated to line up its timestamps with predicted trajectory **x**.

```
measurement_cov = np.array(
    [
        [0.5, 0.2],
        [0.2, 0.5],
    ]
)
control_std = 0.5
factors = (10, 1, 0.01, 0.001, 0.0001, 0.1)
particle_counts = (10, 50, 200, 1000)
```

Fig 8.2 - In this analysis, each combination of **(measurement covariance matrix, control standard deviation, particle count**), such that *measurement_cov* and *control_std* were multiplied by *factors* shown below, was run automatically. This took around 5 hours to run.

Using this metric, a comparative analysis of many combinations of noise parameters + particle count was run (See Fig 8.2). This initial analysis focused on order-of-magnitude for the noise parameters; for the control standard deviation, which is a scalar, this requires no explanation. For the measurement covariance matrix, the base matrix was selected empirically after a variety of experiments with 50 particles.
This sort of covariance makes a rough sort of sense if the noise from bearing & range were somewhat independent of each other, but not entirely so (since the entries on the diagonal represent how the bearing & range of the landmarks suffer from independent noise, and those on the top right & bottom left represent how they suffer from noise in the joint distributions with each other).

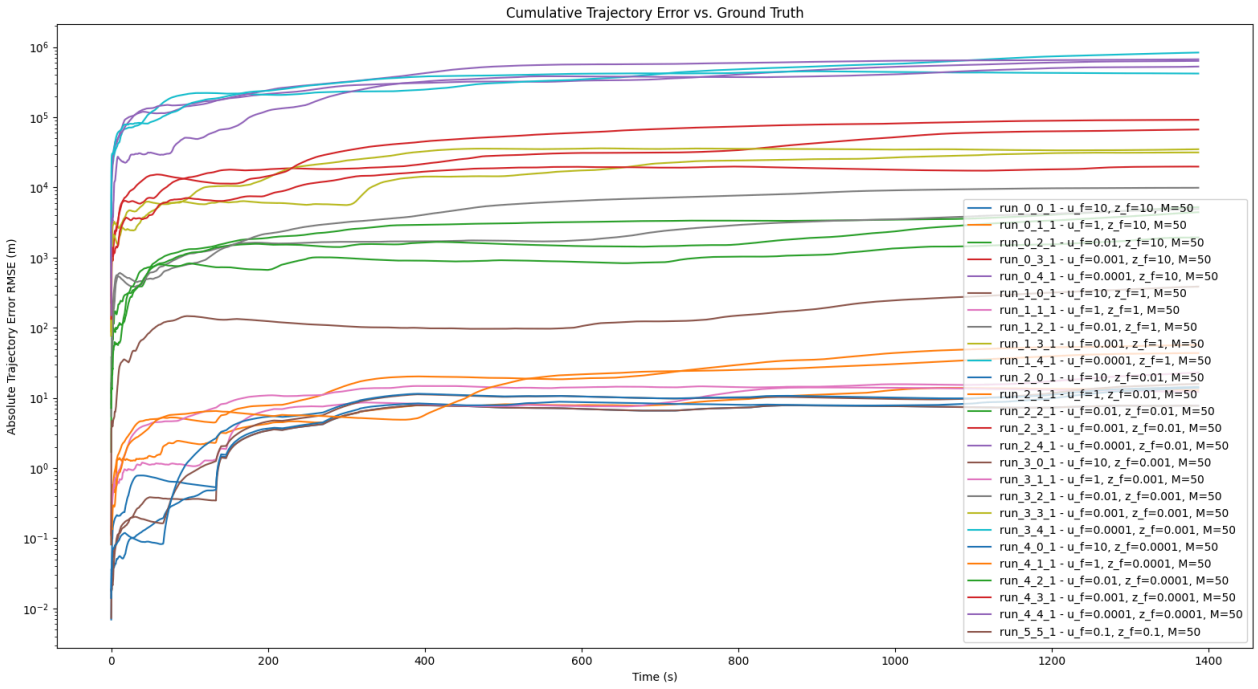

Fig 8.3 - large-scale analysis using the combinations of parameters shown in Fig 8.2. It is clear in this plot that 3 distinct groups appear, corresponding to the order of magnitude of the control noise standard distribution.

After this first exploratory analysis, this lowest-error grouping was further explored, ultimately resulting the selection of noise parameters shown in Fig 8.3 and the trajectory followed in Fig 8.1

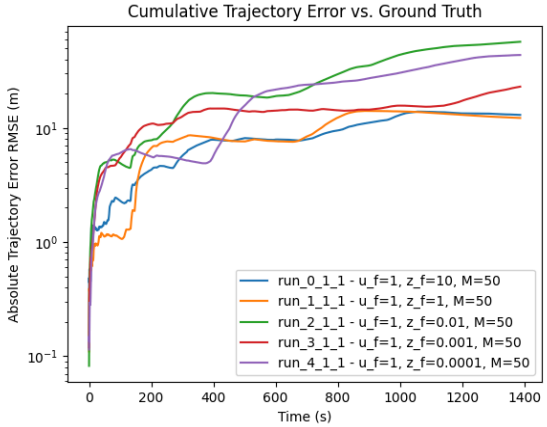

Fig 8.4 - selecting only the u_stddev = 0.05 group from Fig 8.3 above

To observe the result of missing measurements (during which the proposal distribution **bel(x_t)**, aka $\mathbf{X_t}$ is used directly, because no measurement corrections are applied) we can show on the following pair of figures that the precipitous jump in error at the start of the trajectory occurs at the times when measurements are missing (t=125, t~145).