Project 3 - Filtering, GTSAM, and Localization

Hoang-Dung Bui,
George Mason University
Fairfax, USA
hbui20@gmu.edu

I. Discrete Filtering and HMMs

A. A simple HMM

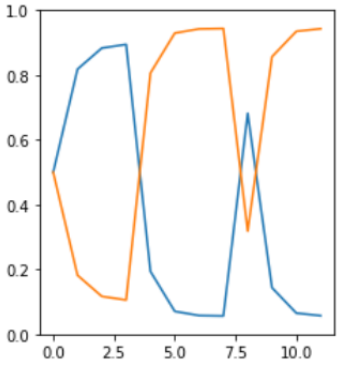


Fig. 1: Simple HMM

TASK: Implement the $predict$ and $update$ functions following the predict and update functions from lecture, and the function hmm_filter function. The result is shown in Fig. 1.

B. Missing some Observations

Question What would happen if you did not receive some observations?

Answer: If some observations are not available, we don't have the data to update the state's probabilities of the system. The more missing of observations, the less accuracy of the system states (which shown by their probabilities). The system's states are predicted only by the motion model.

Sketch: Draw a graphical model corresponding to three steps each with an observation, a step in which no observation was received, and 2 more steps with observations.

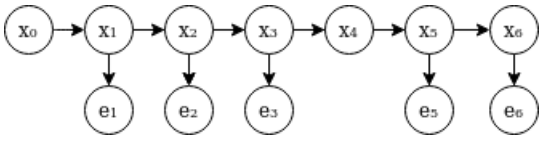


Fig. 2: Missing 1 observation

PLOTS Run the plotting code with $None$ values, and the results shown in Fig. 3.

QUESTION What happens after receiving multiple [None] observations? What values do the probabilities approach (and why)?

Answer: The graph show the probability of system's state over time step. If there are [None] observation, we cannot update the system's state by the $update()$ function, which results is low accuracy for system state prediction. As a result, the probabilities approach 0.5, which means we are not sure what is the current state of the system.

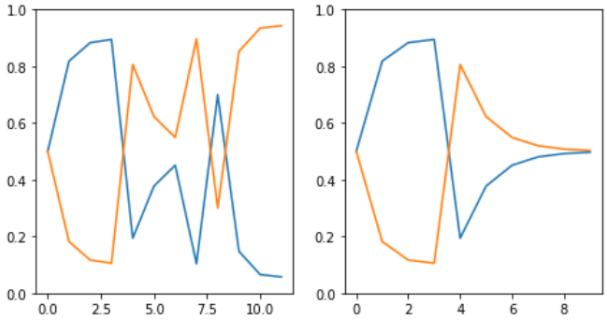


Fig. 3: Missing some observations' result

II. The Kalman Filter

TASK: Implement the $kalman\_filter$ function. The code of Kalman Filter is shown in Fig. 4. First, we use the motion model to update the new position of the robot, and update the covariance matrix. We then calculate the Kalman filter gain K, and use it with the GPS data to update the positions of the robot. The position and the covariance matrix are saved for the next time step.

```
last_x = x0
last_y = y0
last_cov = cov_prior
count = 0
for (dx, dy, xg, yg) in zip(dx_noisy, dy_noisy, x_gps, y_gps):
    # Prediction
    x_new = last_x + dx
    y_new = last_y + dy
    new_cov = last_cov + cov_motion
    # Kalman filter gain
    K = new_cov @ np.linalg.inv(new_cov + cov_gps)
    if do_use_gps:
        # Update Step
        x_new = x_new + K[0,0]*(xg-x_new)
        y_new = y_new + K[1,1]*(yg-y_new)
        new_cov = (np.diag([1, 1]) - K) @ new_cov
    positions.append([x_new, y_new])
    covariances.append(new_cov)
    # save for the next time step
    last_x = x_new
    last_y = y_new
    last_cov = new_cov
```

Fig. 4: The code of Kalman Filter

PLOTS We ran the Kalman filter function and plotted the results of both without and with GPS in Fig. 5. There was

significant difference in the result. Without GPS, the robot's position estimate is worse and worse by time step, and its variance gets larger and larger. With GPS signal, we can update the position's mean and distribution covariances, so the mean is close the the true value and the variance is small and stable over time.

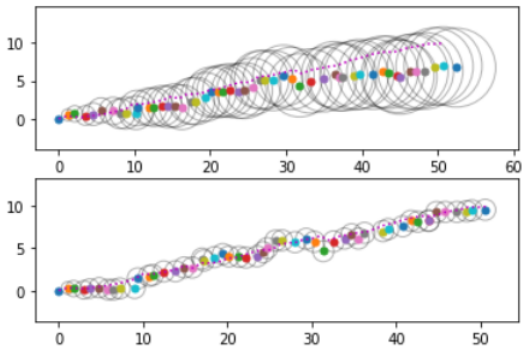


Fig. 5: The Kalman Filter's Result

III. GTSAM

A. GTSAM Fundamentals

PLOT In the absence of any GPS signal, noise compounds whenever the robot moves and the error grows larger and larger as shown in Fig. 6.

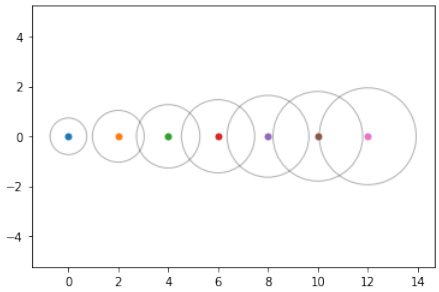


Fig. 6: GTSAM without GPS

TASK Add a GPS signal! At point 6, the GPS tells you that the robot is at point (10.02, 0.0). The position of the robot with GPS signal is much better, close the true value and the covariance is small. As close to the know position (initial position and position 6), the accuracy is high.

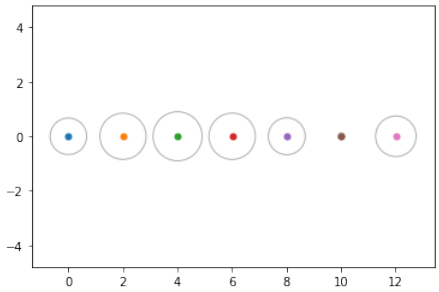


Fig. 7: GTSAM with GPS

QUESTION What happens to the covariances? In particular, comment on what happens to the covariance ellipse at point 5 (and why)?

Answer: The covariance represents the size of the robot position's distribution. It means the true position of the robot can lie anywhere in the covariance circles. With GPS signal, we can update and improve the covariance value and make our prediction close to the true value.

At point 6, we know the position of robot by GPS signal. GTSAM uses this accurate data to smoothing the previous positions of the robot. The closer the points to the GPS position, the higher accuracy the points can get. Therefore, the covariance at point 5 reduces most, then goes to point 4 and 3.

B. GTSAM Version of 3.2 Example

TASK Use GTSAM to build a factor graph using the noisy GPS data from P3.2. The added code is shown in Fig. 8.

```
# Now you do the rest!
for i in range(len(dx_noisy)):
    graph.add(gtsam.BetweenFactorPoint2(i, i+1,
            gtsam.Point2(dx_noisy[i], dy_noisy[i]), ODOMETRY_NOISE))
    graph.add(gtsam.PriorFactorPoint2(i+1,
            gtsam.Point2(x_gps[i+1], y_gps[i+1]), GPS_NOISE))
    initial_estimate.insert(i,
            gtsam.Point2(x_noisy[i]+np.random.normal(0.0, 0.3),
                         y_noisy[i]+np.random.normal(0.0, 0.3)))
initial_estimate.insert(len(dx_noisy),
            gtsam.Point2(x_noisy[-1]+np.random.normal(0.0, 0.3),
                         y_noisy[-1]+np.random.normal(0.0, 0.3)))
```

Fig. 8: GTSAM Extend

PLOT The result of the GTSAM example is shown in Fig. 9. We can see, with GPS signal, over 50 time steps, the robot position estimate is still close to the true ones.

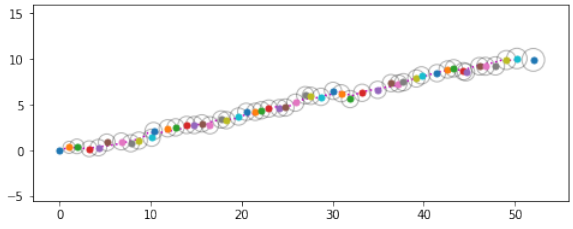


Fig. 9: The result of GTSAM Extend

QUESTION You may notice that the GTSAM example is quite similar to your Kalman filter example, but will be slightly different. Describe how they are similar and different. Why are there differences? What is GTSAM doing that the Kalman filter example does not?

Answer: There is slight difference between Kalman filter and GTSAM's results. The GTSAM's results is smoother and the covariance is smaller. The reason for that is, in Kalman filter, it uses the GPS data to calculate the forward step, it means it process the current step and the future one. Meanwhile, GTSAM uses the GPS data for smoothing, it means it does not only process the current step, but also

handles the previous steps (the past steps). As it gets GPS signal, it both improve the covariance of the current step and the previous steps. Therefore, its trajectory and covariance are smoother and smaller than Kalman filter.

C. GTSAM for Different Drive

TASK The code of function $compute\_updated\_states$ is shown in Fig. 10. After running the plotting code, we have two figures without GPS and with GPS signal in Fig. 11 and 12.

```
if observation['motion'] is not None:
    motion = observation['motion']
    if ii > 0:
        graph.add(gtsam.gtsam.BetweenFactorPose2(ii-1, ii,
          gtsam.Pose2(motion[0], motion[1], motion[2]), ODOMETRY_NOISE))

if do_use_gps and observation['gps'] is not None:
    gps = observation['gps']
    # NOTE: the 'GPS' also has angle (for simplicity)
    if gps is not None:
        graph.add(gtsam.PriorFactorPose2(ii,
                        gtsam.Pose2(gps[0], gps[1], gps[2]), GPS_NOISE))
```

Fig. 10: Function compute_updated_states

QUESTION Describe what you see both without the GPS and with it. Without the GPS, what happens to the covariance matrices? Why are they not "circular" as they have been?

Answer: The common thing between the two figures are the significant difference between the true trajectories of the robot and the estimated trajectories that provided by the motion model. There are so much noise in the motion model, so with or without GPS signal, it is impossible to drive the robot to the goal with open-loop control.

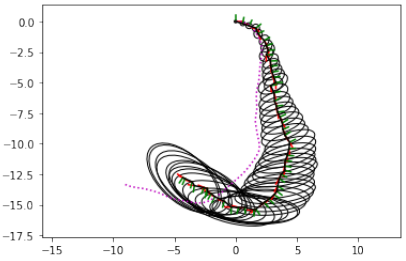


Fig. 11: Position's Estimation without GPS

However, with GPS signal, the covariance of the position's distribution is update in each time step, so it remains in small, and the ellipse's size are stable during the time. The covariance is not "circle" because the wheels' error and rotating angle errors are significant. So, each time the robot makes a turn, the errors in x and y direction are not the same, and it makes the covariance's shape turning to ellipses.

D. Filtering and a Controller

TASK+PLOT Complete the function of the simple diff drive controller and the result is shown in Fig. 13. The black line - the estimate of the robot position - heads towards the goal (the blue star), while the true position (magenta line) is faraway to the goal. It is due to the terrible malfunction of the motion

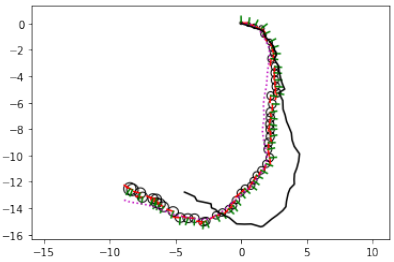


Fig. 12: Position's Estimation with GPS

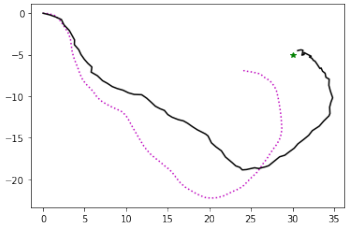


Fig. 13: Controller without GPS signal

model. If we rely on the odometry data, even a closed-loop controller cannot handle it.

TASK+PLOT Compute the true robot state using both the motion model and GPS measure by using the function $compute\_updated\_states$, we can improve much of the robot's position accuracy (shown in Fig. 14). Although the robot motion model provides terrible data, by using the GPS data and smoothing by GTSAM, the robot still reaches the goal (the dashed magenta line).

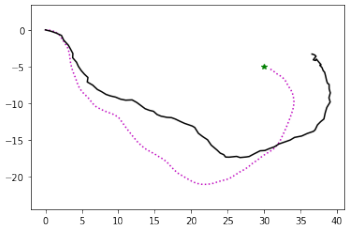


Fig. 14: Controller with GPS signal

IV. Particle Filter Localization

A. Designing an Observation Model

TASK Complete the function $prob\_of\_scan$ function.

```
For <Pose x:150.0, y:150.0, yaw:0.0>: likelihood=1.0
For <Pose x:152.0, y:150.0, yaw:0.0>: likelihood=0.72
For <Pose x:150.0, y:250.0, yaw:0.0>: likelihood=0.14
For <Pose x:120.0, y:650.0, yaw:0.0>: likelihood=2.02
For <Pose x:180.0, y:750.0, yaw:0.0>: likelihood=0.00
```

Fig. 15: The Result of $prob\_of\_scan$ function

RESULT Run the "results" code and the results are shown in Fig. 15. The likelihood of the "true" pose is very good and decreases for scans that look completely different.

B. Computing Particles

TASK Complete *initialize_particles*, as defined below. Create a set of *num_particles* Particles, each with a random starting pose (and fixed probability). The code is shown in Fig. 16.

```
for _ in range(num_particles):
    pose = get_random_pose(grid)
    particle = Particle(pose, 1/num_particles)
    particles.append(particle)
return particles
```

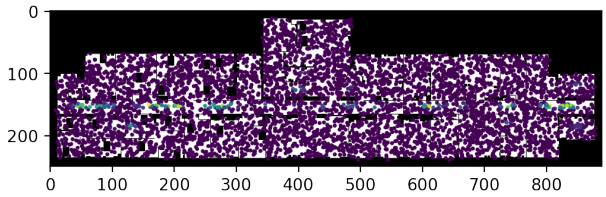
Fig. 16: The function *initialize_particles*

TASK Complete the *move_robot_and_update_particles* function. This function first moves the robot and generates a scan (observation). It then moves each of the particles using *particle.move* and the known motion of the robot and updates the weight of each of the particles using the *prob_of_scan* function you used above. The code detail is shown in Fig. 17.

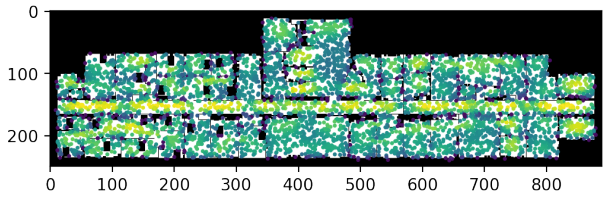
```
for particle in particles:
    particle.move(dx, dy, dyaw)
    sim_ranges = laser.simulate_sensor_measurement(
        grid, directions, max_laser_range, particle.get_pose())
    particle.prob = prob_of_scan(observed_ranges, sim_ranges)
```

Fig. 17: The function *move_robot_and_update_particles*

PLOT Run the plotting code and the results are shown in Fig. 18a and 18b. The brightness of each point corresponds to its likelihood. Most of the bright spots in the hallway, since those measurements are similar to the observed scan from the true robot pose.



(a) The particles after the first move without log-scale



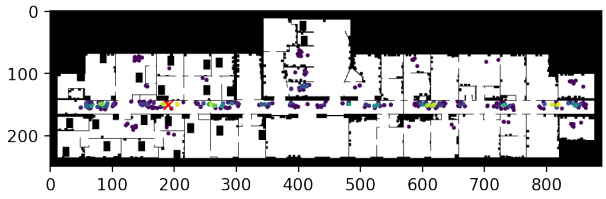
(b) The particles after the first move with log-scale

Fig. 18: The particles after the first processing step

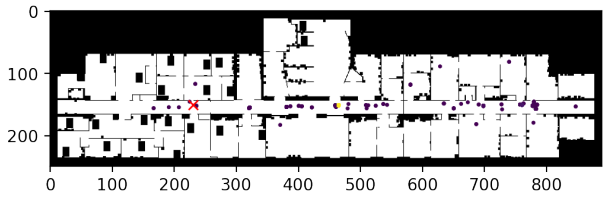
C. Resampling and Localizing

TASK Implement the *resample_particles* function below using the procedure described above.

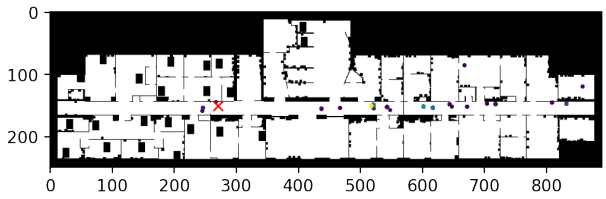
PLOTS One result from the code is shown in Fig. 19a 19b, 19c and 19d. For this time, the algorithm can determine quite accurate the true position of the robot, and the brightest points are displayed close to the robot true position.



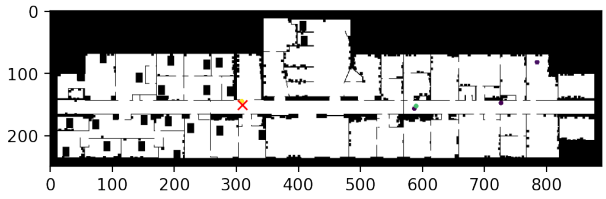
(a) After iteration 1



(b) After iteration 2



(c) After iteration 3



(d) After iteration 4

QUESTION What is the ”results” code doing? How is the robot moving. Is our particle filter converging on the right positions over time?

Answer: The code works, however it is not stable. Sometimes, it can estimate correctly the position of the robot after 4 iterations as showing in Fig. 19a 19b, 19c and 19d. But for another iterations, it works badly, and estimates faraway from the true position. I think the instability is due to the similarity among multiple position in the maps. If it estimate wrong in the first iteration, as a consequence, it will work worse by later iteration.