Corentin Chauvin-Hameau

KTH Formula Student – SLAM task

https://github.com/CorentinChauvin/sensor_fusion

Basic Concepts Quiz

1. What is odometry?
→ A measurement of relative movement. Often it is done by measuring the rotation of the wheels with coders, it can also be by comparing images (visual odometry).

2. What are the main problems related with odometry?
→ The position is given by integrating the speed measured. It means the small measurement errors made will accumulate. Moreover, if the odometry is given by coders directly on the motorised wheels, it won't take into account potential drifts.

3. Can odometry can be used as the only mean of localization? Why?
→ It depends on the level of precision needed and the distances. For high precision or big distances, the precision will become to bad without any other mean of localization.

4. What is dead reckoning?
→ It is relying only on proprioceptive measurement, without sensing the environment.

5. What are the main challenges when applying an Extended Kalman filter?
→ When implementing the filter, one need to determine the equations that link the state to the measurements. Then, the trickiest part is to tune the different covariance matrices, in order to find the good balance in the importance given to the different measurements.

6. Define the SLAM problem with you own words.
→ It is about building a map while determining one's position at the same time.

7. How do occupancy grid work?
→They subdivide the space in cells, and for each cell, they give a probability of presence of an obstacle inside it. With it, one can estimate what are the free and occupied spaces.

8. What are the main pros and cons of a Particle Filter for navigation?
→ It can give a good representation of any probability distribution (and especially non gaussian one). But its complexity grows very fast with the dimensions, so it is very bad to estimate complex states (such as maps for instance).

9. What are the main challenges in robotics navigation?
→ One of the big challenge I can think about is the movement. It is nowadays hard to build maps in dynamic environment, where objects or people move around. And for pure navigation, this is also a problem. How to plan a trajectory when there could be things or people bumping into it? One need to understand the way objects/people could behave in the future seconds…

10. What is a holonomic robot.
→ It is a robot which can move without the non-slipping constraints. A wheeled robot equipped with swedish wheels is a good example.

Development of a localization solution

My work is mainly based on the ROS *robot_localization* package. It is an amazing package when it comes to fuse odometry or IMU data. It can also deal with GNSS data or visual odometry as long as they are converted in the right format (an odometry message).
The estimated state estimated is the 6D pose of the robot. The prediction step is done by integrating a full dynamical model taking into account pose, twist and accelerations. The update step fuse all the sensors measurements available at the time.
The package is highly tunable. For instance, we can easily decide what data fuse for each sensor (angular speed along x, linear acceleration along z, x position…).

What I did first was to understand the structure of the data set, read it, and publish ROS messages with it. For each sensor (IMU, speedometer, and GNSS), I build a list of ROS messages (sensor_msgs/Imu for IMU and nav_msgs/Odom for speedometer and GNSS). Then, in the main loop, I compare the ROS time to the headers of the messages and publish it if their time came. With that, everything is published as if the measurements were taken in real time.
Since the position of the GNSS is given in NED frame (North, East, Down), the data of the GNSS is absolute, but not the one given by the odometry. To get an estimation of the absolute heading, I determined it by looking at two consecutive positions of the GNSS.

Then, I needed to estimate the different covariances of the measurements, to use it in the Kalman filters. For the GNSS and IMU, I took the precisions provided for the Xsensx IMU+GNSS unit. For the speedometer, without any reliable data, I defined the covariance as a percentage of the speed (we want it to be proportional to the speed since the errors are bigger with high speed).
If the distance between two different GNSS positions was too close, I set an arbitrary high covariance for the heading, since it would be very very bad given the imprecision of the GNSS. Else, the covariance is given by differentiating the expression that gives the heading (see the code for the details).

With all the data ready to be processed, I set up a common robot_localization structure: using two different kalman filters (here, Unscented Kalman Filters). The first one is fusing the linear speed in x of the odometry (the speedometer value so) and the angular velocity along z of the IMU. This produces an odometry in the *odom* frame.
A second UKF fuses linear speed along x and angular speed along z of the odometry given by the first filter, and also the pose given by the GNSS. The output of the second filer is a new odometry given in frame *map*.
These frames are important, we get a tf tree looking like $map \rightarrow odom \rightarrow base\_link$. The position of the robot will be continuous in *odom*, but will drift in time. The position in *map* will not drift, but will jump at each update.

Finally, the trickiest part as I said in the previous page: tuning the process noise covariance of the kalman filters (in robot_localization.yaml). It is done by balancing the influence of each measurement on each state component.

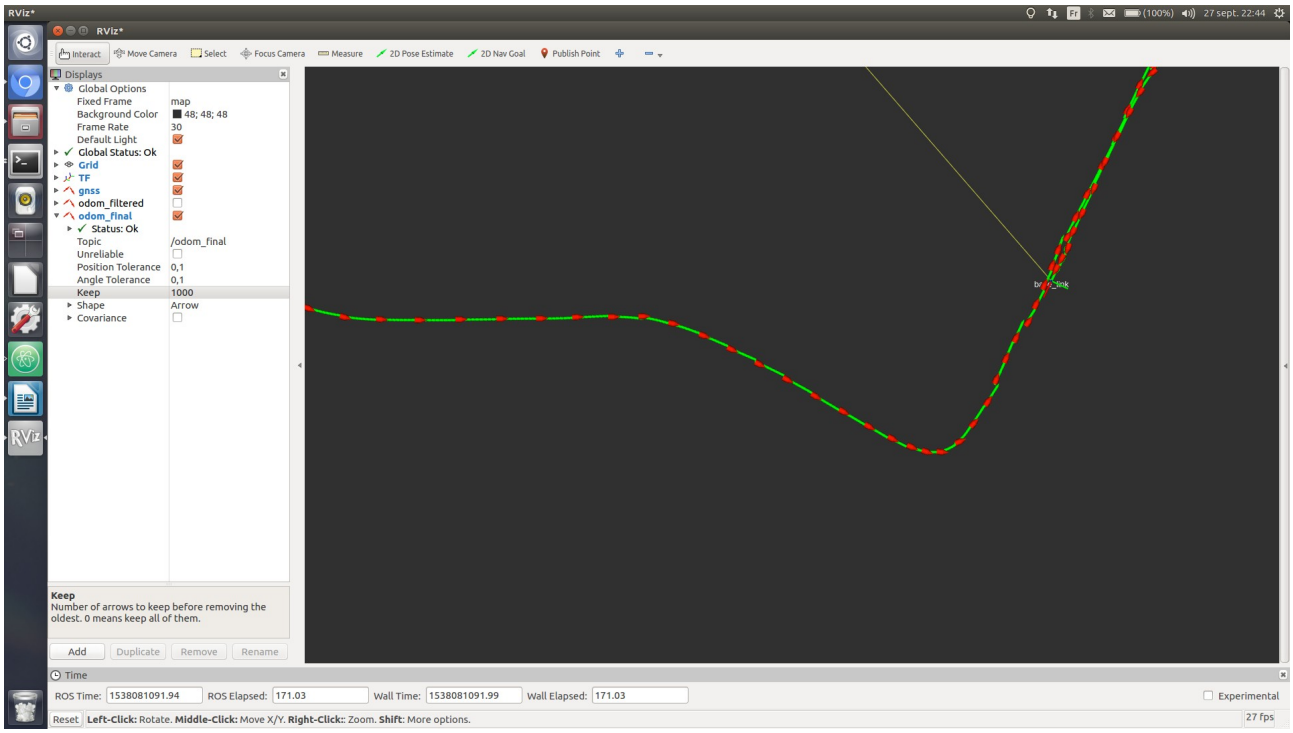
What needs to be done to improve it: at first have a look at the real behaviour of the car. One needs to see what we can expect of the different data. Is the GNSS as precise? Is there lots of slipage, and so the odometry is to be considered with caution? etc.
Then, better tune the filters accordingly.

Questions about the task

1. What can happen if 1 measurement is delayed?
→ The magic in robot_localization is that it takes into account the stamps of the sensors data. This means that at each update step, it processes the measurements available in chronological order. Thus, as soon as a delayed measurement is provided to the filters, the information it contains is computed in the past to update the state in the present.

2. Now your IMU gets damaged. How your implementation deals with it?
→ The position between each GNSS update will be very bad. Indeed, the first filter will fuse non sense data. But in the second filter, the GNSS will correct the position, and depending on the tuning, it could prevent it from diverging. Moreover, if the covariances provided by the IMU (which I should use instead of constant covariances) are aware of the damages, it could contain the divergence.

3. How will you deal with the case when the car is in a place where it has been before, but there is an error and localization is showing a different place?
→ I am not fully sure to understand the question, but if the localization is suddenly at a completely different place from where it actually is, I would be upset by the data given by the GNSS. However, if only the initial position is wrong, the filters will converge to the real position thanks to the absolute measurement of the GNSS. It will take a certain time, depending on the tuning, but it will converge.



In red, GNSS. In green, final output of the second kalman filter.