Navigation strategies for uncertain sensory data

Dhruv Maroo

April 23, 2021

Problem statement

A typical architecture for a model-based navigation system involves 4 major components - Perception, Localisation, Planning, and Motion Control. Consider our VIRAT. It has sensors that determine its position and motion, as well as a model to predict its future positions. In the real world, predictive models and sensors aren't perfect. There is always some uncertainty. For example, the weather can affect the incoming sensory data, so the bot can't completely trust the information. Suggest a way on how you would refine the robot's navigation stack to mitigate this uncertainty.

Possible solutions

I list the possible solutions that I could think of in the following paragraphs. These solutions could be implemented alongside each other, or may be used as standalone solutions in certain scenarios.

Checking directly derived data with indirectly derived data

Let us suppose that our robot has a **odometer** and a very accurate **positioning sensor**. We can directly derive the *linear velocity* of our robot using the data supplied by the odometer. Let us call this velocity as `lin_vel_direct`. Also, we can use the position data we received to calculate the velocity, inexpensively (without using much time or memory). Let us call the velocity calculated indirectly using the position data as `lin_vel_indirect`.

Now we can check whether both, `lin_vel_direct` and `lin_vel_indirect`, are close enough or not. If they differ significantly, then we can be sure that at least one of the sensors has not provided accurate data. In such a case, we can trigger a "safe" and "restrictive" motion algorithm, where the robot either slows down or chooses a path it already knows for sure, to continue it's motion. This way, we can wait until both `lin_vel_direct` and `lin_vel_indirect` get close enough, which will (mostly) signify that we are getting somewhat accurate data and we can trust that data and navigate accordingly.

Accuracy of the above procedure

The above procedural algorithm has very less chances of giving false positives. This is because, to create a false positives, data from both the sensors needs to be inaccurate, and in such a way that `lin_vel_direct` and `lin_vel_indirect` are also close enough. This is only possible in the case of a rare coincidence. Therefore, this algorithm will work well. Also, we can extend this algorithm to other parameters (apart from just velocity) as well. As long as there are more than one ways to derive the value of a quantity, we can apply the above procedure.

Using post-processing on the raw data and changing the threshold according to the situation

However, in the cases where there are very frequent disturbances in the sensory data, this algorithm will slow down the robot very frequently, thus we will have to post-process the raw sensory data to remove the noise, and increase the difference threshold between `lin_vel_direct` and `lin_vel_indirect`, so that the robot only pauses if there is a big enough difference between both the velocities. This way, we can overcome the intermittent slowing down of the robot.

Checking the coherence of the received data

As we did above, we can similarly check whether the data we received from the different sensors is coherent or not. If it isn't coherent, then we will reduce the weight of the incoherent data element while calculating the motion.

Constantly monitoring received data and cached data

Whenever we instruct the robot to move, we can simultaneously update the current parameters of robot (which we could store in a cache) according to our instructions, and when we receive the next blob of sensory data from the robot, we can compare that data with the stored data in the cache. If the data which we received differs significantly from the calculated data, then we know with a high certainty that one or more sensors returned inaccurate data. We can also pinpoint the sensors by comparing and checking every single parameter of the raw data received. This way we can filter out all the uncertain sensory data, compute the further instructions, transmit them back to the robot, and simultaneously update the cached data. This loop will go on and on as long as the robot is in motion, allowing us to mitigate the problems that inaccurate data can cause.