Record For Implementing UKF on Existing Round Move

Mingze Chen

April 21, 2021

1 Implementing UKF

After connecting to the robot through *ssh*, then execute the following command:

```
husarion@husarion: $ roscore
```

start 2nd. command line window and execute:

```
$ roslaunch rosbot_ekf all.launch rosbot_pro:=true
```

start 3rd. command line window and launch the *robot_localization* through executing:

```
~/pathTo/catkin_ws$ source ./devel/setup.bash
~/pathTo/catkin_ws$ roslaunch playground start_filter.launch
```

Now our purpose is implementing *UKF* algorithm into existing *round_move.py* script.

To launch the *UWB tag* through executing:

```
~/pathTo/catkin_ws$ source ./devel/setup.bash
~/pathTo/catkin_ws$ roslaunch localizer_dwm1001 dwm1001.launch
```

inside the *playground* package there's a **Python** script called *round_move_ukf.py*. Now execute:

```
$ husarion@husarion:~/pathTo/catkin_ws$ rosrun playground
    round_move_ukf.py
```

The running process was: we let the robot to move back and forth twice with single distance of 2m, which makes a total distance of 8m, we repeat twice with the same starting position for the robot, the difference is: Before the 2nd. run we calibrated the coordinate of UWB more precisely with *BOSCH Laser* *measure* but only 3 stabel working anchors while 4 stable working anchors with worse coordinates precision in the 1st. run.

Afterwards, position data based on calculation and *UKF* are collected separatedly in files called *pose1204.csv*(1st. run) and *pose2104.csv*(2nd. run) after plotting, we get Fig. 1 and Fig. 2:

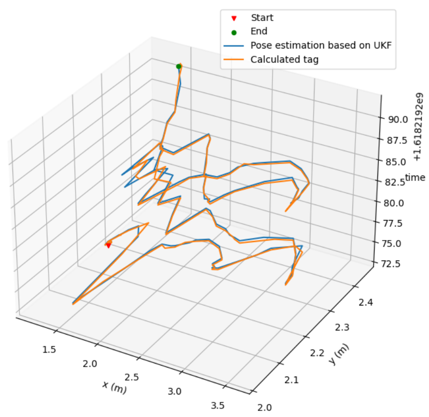

Figure 1: UKF vs. UWB alone

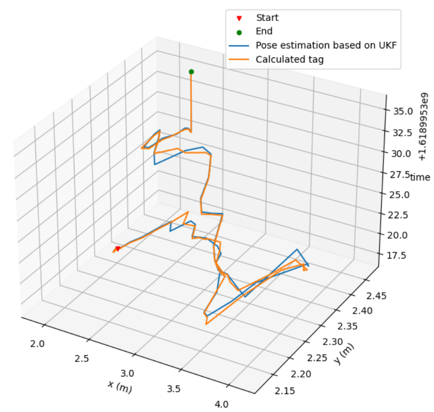

Figure 2: UKF with UWB calibration and 3 stable working anchors only vs. UWB alone

The *khaki trace* denotes the Cartesian coordinates based on *UKF* calculation, while the *blue trace* represents the raw *UWB tag* position, which is obviously not very stable.

Our calculation process is:

1. Initial position was calculated based on *UWB*
2. Afterwards, every small step was calculated based on an *internal EKF* with only *IMU* and *Odometry* as input and an external *UKF* with all sensor inputs

At the same time, *mean square error of each run was calculated* through *Python* script `MSE_cal.py` and the calculation result was:

```
$ MSE for 1st. run = 0.0014062992125984232
$ MSE for 2nd. run = 0.0003471074380165288
```

So from the comparison we can see that the calibration made to UWB anchors has brought us some measurement improvement, although far from enough.

2 Idea for next step

1. Try to get the orientation of the robot, e.g. from quaternion (subscribe from topic *mpu9250*) to euler, which can be extremely useful in the process which involves turning
2. Measure the process error through observation: Let the robot move forward with a velocity of 0.5 m/s and record its relative position after 1s, repeat e.g. 1000 times, which makes it reliable
3. Try to read IMU data from UWB tag sensor unit
4. Fuse uwb tag *pos* + IMU to eliminate *pos* errors
5. Conducting repeated experiment and at the same time, gather raw sensor data, which contains more noise definitely and try to reduce the noise
6. Consider using Graph Neural Networks, *Particle Filter*, not only UKF and EKF
7. Consider designing a random walk model for the final demonstration

3 Results and Conclusions

From both Fig. 1 and Fig. 2 we can see, after conducting *UKF*, *tag* was closely followed, so the precision of the tag matters quite a lot. However, the actual movement was along the x-axis, the change in y-axis should be no more than 5cm theoretically, which is maximum around 30cm in the Figure, should be eliminated.