Record For Integrating UWB Data Into ROS

Mingze Chen

March 25, 2021

1 Integrating UWB

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 integrating UWB data into existing *move.py* script, there're at least 2 possibilities:

- Execute in a multi-threading/multiprocessing way
- Publish UWB data into a ROS topic and subscribe when- and wherever needed

For the moment I've chosen the 2nd. method because there's no real multi-threading in Python, here's the explanation:

"The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter. This means that only one thread can be in a state of execution at any point in time."

If we bypass this through writing some *C* code, it would be complicated. As for multiprocessing mechanism, which will bring relative bigger change to the existing code base. So the classical *publisher and subscriber* model is preferred here.

There is already package support for *Decawave DWM1001C [Dec21]* called localizer_dwm1001. After cloning this package into workspace in src folder /pathTo/catkin_ws/src/ and execute:

```
$ catkin_make
```

Now execute following command we can check running rostopic:

```
$ rostopic list
```

And from the output we can know */dwm1001/tag* is exactly what we want and through executing:

```
$ rostopic type /dwm1001/tag
```

we can know the message type is *localizer_dwm1001/Tag*, which we will use later in order to subscribe from the topic and decode the coordinate message.

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 *move_uwb.py*. Now execute:

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

In the running process we told the robot to move forward for 2m.

Afterwards, position data based on calculation and *UWB sensor unit* are collected in a file called *poseRecord1832021.csv* and after plotting, we get figure 1:

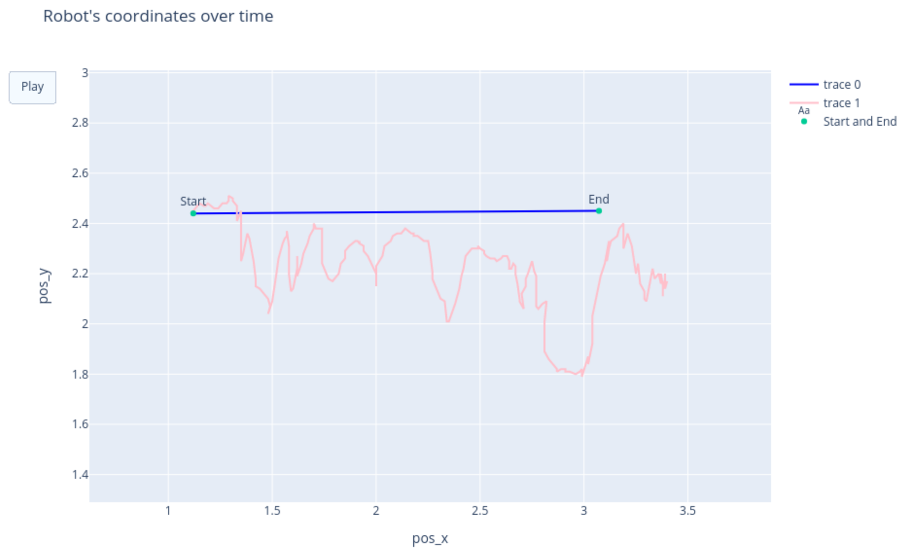

Figure 1: Demonstration of running process of the robot

trace 0 denotes the coordinates based on our calculation, while *trace 1* represents the raw *UWB* data, which is obviously not very stable.

Another test was also conducted and data was collected in *poseWithTime.csv* and was 3D plotted through *3Dplot.py*, as shown in figure 2:

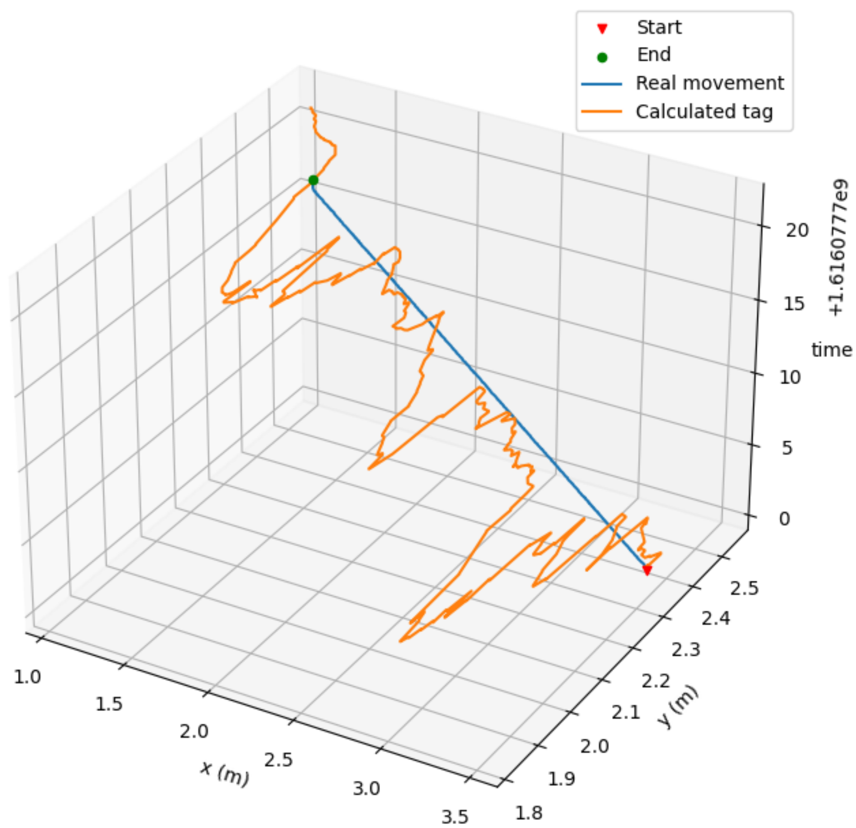

Figure 2: Demonstration of running process of the robot in 3D

Our calculation process is:

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

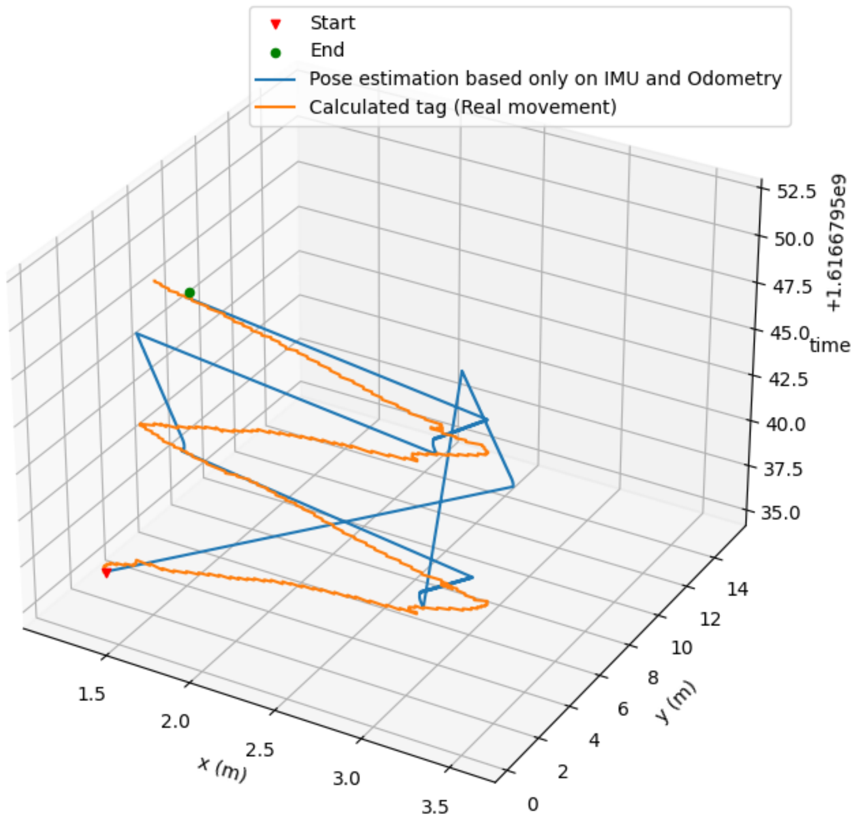

Figure 3: Round movement illustration

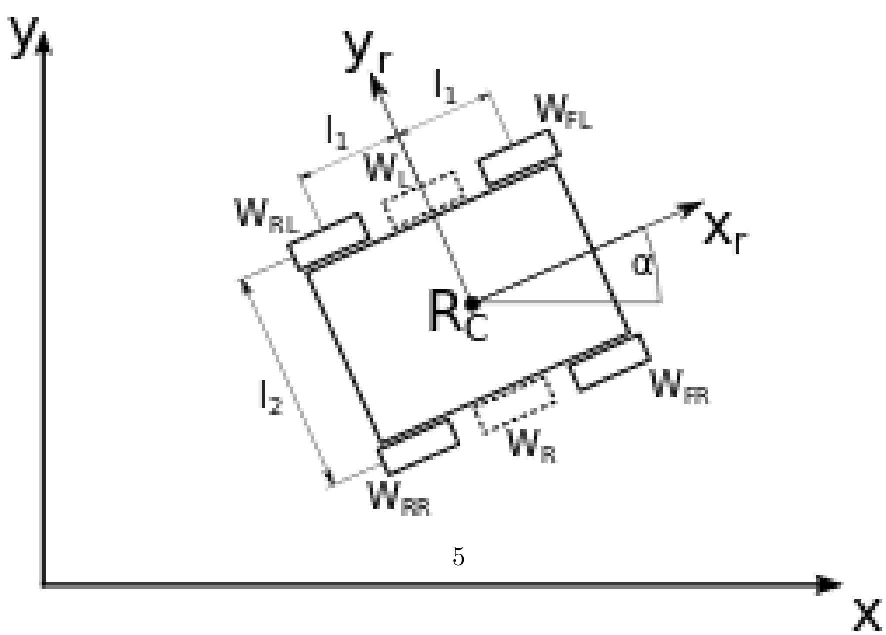

Figure 4: Robot Scheme[1]

2 Idea for next step

1. Single experiment is not that meaningful, run repeated experiment and at the same time, gather raw sensor data, which contains more noise definitely and try to reduce the noise

   1.1. let the robot move a perfect square, which is proven by Python script *square_move.py* to be quite hard to turn exactly 90°

   1.2. move the robot back and forth twice with 2 m for every single movement, which makes the robot travel a total distance of 8 m, as shown in figure 3 and the error was measured manually: the robot should come back to (0, 0) where it originally started, but according to figure 4 the real position was at around (0.24, -0.075) (unit: meter), as shown in figure 5.

2. Consider using Graph Neural Networks, not only EKF

3. Consider designing a random walk model for the final demonstration

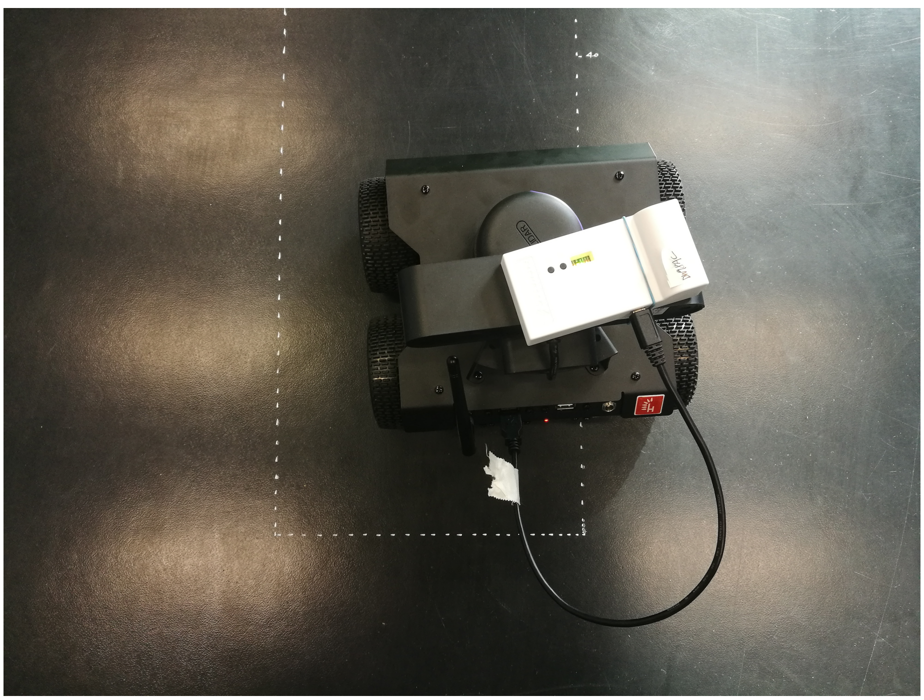

Figure 5: The actual stop position after round movement

[1] https://husarion.com/tutorials/ros-tutorials/3-simple-kinematics-for-mobile-robot/

3 Results and Conclusions

Compare figure 2 with figure 3 we can know: within a short movement, e.g. 2m, EKF based on only IMU and odometry can be quite accurate. However, as the distance grows, like 8m in total as shown in figure 3, UWB begins to play a vital role, through which we can also infer our pose estimation algorithm should heavily rely on *UWB*.

References

[Dec21] Decawave. *DWM1001C Module*. 2021. URL: https://www.decawave.com/product/dwm1001-module/ (visited on 02/12/2021).