Record For Gathered Odometry Data and fusing it with IMU data

Mingze CHEN

June 4, 2021

1 Gathered Odometry Data Form

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

```
1    husarion@husarion:~/pathTo/catkin_ws$ roscore
```

start 2nd. command line window and execute:

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

Now we can check the *Odometry* data through executing:

```
1    $ rostopic echo -n1 /odom
```

The output from the command line is:

```
1    header:
2    seq: 786
3    stamp:
4      secs: 1614863050
5      nsecs: 814419031
6    frame_id: "odom"
7    child_frame_id: "base_link"
8    pose:
9      pose:
10       position:
11         x: 0.0
12         y: 0.0
13         z: 0.0
14       orientation:
15         x: 0.0
16         y: 0.0
17         z: 7.19295913417e-05
18         w: 0.999999997413
```

```
19    covariance: [0.00479857518885908, −1.796532892888607e−24, 0.0,
          0.0, 0.0, 0.0, 1.822382287030889e−24, 0.00479857518885908,
          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.005351925638176956, 0.0, 0.0,
           0.0, 0.0, 0.0, 0.0, 203.74663407959122, 9.540979117872439e
          −18, 0.0, 0.0, 0.0, 0.0, −9.540979117872439e−18,
          203.74663407959122, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
          0.004863059421148123]
20  twist:
21    twist:
22      linear:
23        x:  0.0
24        y:  0.0
25        z:  0.0
26      angular:
27        x:  0.0
28        y:  0.0
29        z:  0.00177780095644
30    covariance: [0.06582955266765285, 4.345428558159674e−22, 0.0,
          0.0, 0.0, 0.0, −4.585440182770763e−23, 0.06582955266765285,
          0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0866602439197627, 0.0, 0.0,
          0.0, 0.0, 0.0, 0.0, 0.3934245000348817, 0.0, 0.0, 0.0, 0.0,
          0.0, 0.0, 0.3934245000348817, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
          0.0033845883432777535]
31  −−−
```

The structure of the rostopic */odom* also signifies the method to get e.g. *pose* and *orientation*:

```
1  def odom_callback(data):
2      # rospy.loginfo(rospy.get_caller_id() + " I heard %s", data)
3      print "pose x = " + str(data.pose.pose.position.x)
4      print "pose y = " + str(data.pose.pose.position.y)
5      print "orientation x = " + str(data.pose.pose.orientation.x)
6      print "orientation y = " + str(data.pose.pose.orientation.y)
7      data = str(data)
8      res = re.findall(r"[−+]?\d*\.\d+|\d+", data)
9      # rospy.loginfo(rospy.get_caller_id() + " x = %s, y = %s", res
            [3], res[4])
```

There's an existing ros package called *playground* and execute following command:

```
1    $ pathTo/playground/mkdir launch
2    $ pathTo/playground/mkdir config
```

Then inside the launch directory create a launch file named *start_filter.launch* with the following content:

```
1    <launch>
2
3      <!−− Run the EKF Localization node −−>
4      <node pkg="robot_localization" type="ekf_localization_node"
          name="ekf_localization">
5        <rosparam command="load" file="$(find playground)/config/
              ekf_localization.yaml"/>
```

```
6        </node>
7
8    </launch>
```

Note:

```
1    $(find playground) is encouraged to use to increase portability
         rather than absolute path.
```

A ROS program called *ekf_localization_node* was launched, in which an EKF was applied, with a configuration file called *ekf_localization.yaml*:

The main part of the file was illustrated as follows:

```
1    #Configuation for robot odometry EKF
2    #
3    frequency: 10
4
5    two_d_mode: true
6
7    publish_tf: false
8
9    # the frames section
10   odom_frame: odom
11   base_link_frame: base_link
12   world_frame: odom
13   map_frame: map
14
15   # the odom0 configuration
16   odom0: /odom
17   odom0_config: [false, false, false,
18                  false, false, false,
19                  true,  true,  false,
20                  false, false, true,
21                  false, false, false,]
22   odom0_differential: false
23
24   # the imu0 configuration: yaw(orientation), angular velocity in Z
         and linear acceleration in X
25   imu0: /imu/data
26   imu0_config: [false, false, false,
27                 false, false, false,
28                 false, false, false,
29                 false, false, true,
30                 true, false, false,]
31   imu0_differential: false
```

odom0_differential is generally for multiple odometry.

As for the variables matrix:

$$
\begin{array}{ccc}
X & Y & Z \\
roll & pitch & yaw \\
X/dt & Y/dt & Z/dt \\
roll/dt & pitch/dt & yaw/dt \\
X/dt2 & Y/dt2 & Z/dt2
\end{array} \tag{1}
$$

here is the explanation:

- X, Y, Z: These are the [x,y,z] coordinates of the robot.

- roll, pitch, yaw: These are the rpy axis, which specify the orientation of the robot.

- X/dt, Y/dt, Z/dt: These are the velocities of the robot.

- roll/dt, pitch/dt, yaw/dt: These are the angular velocities of the robot.

- X/dt^2, Y/dt^2, Z/dt^2: These are the linear accelerations of the robot.

So odometry will take *linear velocities in X and Y, and angular velocity in Z* into consideration, and *IMU* cares *yaw(orientation), angular velocity in Z and linear acceleration in X*, which explains the aforementioned matrices.

As for *Covariance matrices*, the parameters in it can be tuned/adjusted dynamically.

After all of these, it's time to launch the *robot_localization* through executing:

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

After this sensor fusion process, two new topics are generated:

- /accel/filtered

- /odometry/filtered

Through executing:

```
1    $ rostopic echo -n1 /accel/filtered
```

the output is:

```
1    header:
2    seq: 15257
3    stamp:
4      secs: 1622812298
5      nsecs: 852361202
6    frame_id: "base_link"
7    accel:
8      accel:
9        linear:
10         x: 0.0
11         y: 0.0
```

4

```
12          z : 0.0
13      angular :
14        x : 0.0
15        y : 0.0
16        z : 0.0
17    covariance : [0.018936834217142526 , 1.1394816767683019e−25, 0.0 ,
            0.0 , 0.0 , 0.0 , 9.01688445236152e−24, 0.018936834217142522 ,
            0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 4.996038034278782e−07, 0.0 ,
            0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 ,
            0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0]
18  −−−
```

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

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

The most obvious benefit from fusing *IMU* and *Odometry* data is: Now the robot can move forward or backward almost 100% correct distance that the program denotes.

2   Idea for next step

   a. Set up *UWB* devices and integrate them into EKF

   b. Consider using Graph Neural Networks, not only EKF

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