Record For Gathered Odometry Data and fusing it with IMU data

Mingze Chen

May 17, 2021

1 Gathered Odometry Data Form

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

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

start 2nd. command line window and execute:

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

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

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

The output from the command line is:

```
  header:
  seq: 786
  stamp:
    secs: 1614863050
    nsecs: 814419031
  frame_id: "odom"
child_frame_id: "base_link"
pose:
  pose:
    position:
      x: 0.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 7.19295913417e-05
      w: 0.999999997413
```

```
  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]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.00177780095644
  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]
---
```

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

```python
def odom_callback(data):
    # rospy.loginfo(rospy.get_caller_id() + " I heard %s", data)
    print "pose x = " + str(data.pose.pose.position.x)
    print "pose y = " + str(data.pose.pose.position.y)
    print "orientation x = " + str(data.pose.pose.orientation.x)
    print "orientation y = " + str(data.pose.pose.orientation.y)
    data = str(data)
    res = re.findall(r"[-+]?\d*\.\d+|\d+", data)
    # 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:

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

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

```xml
  <launch>

    <!-- Run the EKF Localization node -->
    <node pkg="robot_localization" type="ekf_localization_node" name="ekf_localization">
```

```
        <rosparam command="load" file="$(find playground)/config/ekf_localization.yaml"/>
    </node>

</launch>
```

Note:

```
  $(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:

```
#Configuation for robot odometry EKF
#
frequency: 10

two_d_mode: true

publish_tf: false

# the frames section
odom_frame: odom
base_link_frame: base_link
world_frame: odom
map_frame: map

# the odom0 configuration
odom0: /odom
odom0_config: [false, false, false,
               false, false, false,
               true,  true,  false,
               false, false, true,
               false, false, false,]
odom0_differential: false

# the imu0 configuration: yaw(orientation), angular velocity in Z and linear acceleration in X
imu0: /imu/data
imu0_config: [false, false, false,
              false, false, false,
              false, false, false,
              false, false, true,
              true, false, false,]
imu0_differential: false
```

odom0_differential is generally for multiple odometry.

As for the variables matrix:

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

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:

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

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

```
$ 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