Homework #5
ECE495
Kazumi Malhan

1. (a) Two files that are present in every ROS package are CMakeList.txt and package.xml package.xml describes the package by specifying which packages it depends on, among other things. CMakeList.txt specifies input commands to CMake when it compiles the ROS workspace.

   (b) One example use case of private node handle is when you know that there will be multiple instance of same node running on ROS. By using private node handle, each topic will be inside the namespace so that they can be differentiated.

   (c) A complete representation of a coordinate frame transformation consists of translation and orientation component.

   (d) YAML file is used to load multiple parameter values. When YAML file is used, one needs to specify the relative path to the YAML file in the launch file with rosparam tag.

   (e) The determinant of a rotation matrix must be equal to +1 in order to keep the vector's magnitude same. When determinant is not equal to +1, it means that magnitude of the vector has changed during the rotation.

2. Decimal Degree = Degree + (Minutes / 60) + (Seconds / 3600)

$30^\circ + (30/60) + (36/3600) = 30.51$

As it's South -30.51

$5 + (40/60) + (40/3600) = 5.6778$

As it's East 5.6778

Therefore,

Latitude -30.51°

Longitude 5.677778°

3. (a) Pitch $(R_y) = \begin{bmatrix} \cos 90 & 0 & \sin 90 \\ 0 & 1 & 0 \\ -\sin 90 & 0 & \cos 90 \end{bmatrix}$

$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$

Yaw $(R_z) = \begin{bmatrix} \cos 135 & -\sin 135 & 0 \\ \sin 135 & \cos 135 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$= \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Rotation Matrix $= R_z R_y R_x = \begin{bmatrix} 0 & -0.707 & -0.707 \\ 0 & -0.707 & 0.707 \\ -1 & 0 & 0 \end{bmatrix}$

roll $(R_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$

$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$\phi = 0^\circ$

$\theta = 90^\circ$

$\psi = 135^\circ$

(b) $x = \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2}$

$= \sin(0)\cos(45)\cos\left(\frac{135}{2}\right) - \cos(0)\sin(45)\sin\left(\frac{135}{2}\right)$

$= 0 - 0.65328$

$x = -0.65328$

4. (a) Rotation Matrix $= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$ where $\alpha = -60°$

$$= \begin{bmatrix} 0.5 & 0.866 & 0 \\ -0.866 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation vector $= T_G - V_G = \begin{bmatrix} 0-10 \\ 0-10 \\ 0-0 \end{bmatrix} = \begin{bmatrix} -10 \\ -10 \\ 0 \end{bmatrix}$

(b) $T_V = R_V^G [T_G - V_G]$

$$R_V^G = \begin{bmatrix} \cos 60 & -\sin 60 & 0 \\ \sin 60 & \cos 60 & 0 \\ 0 & 0 & 1 \end{bmatrix}^T = \begin{bmatrix} 0.5 & 0.866 & 0 \\ -0.866 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_V = \begin{bmatrix} 0.5 & 0.866 & 0 \\ -0.866 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10-10 \\ 20-10 \\ 0-0 \end{bmatrix} = \begin{bmatrix} 8.66 \\ 5 \\ 0 \end{bmatrix}$$

Target $= (8.66m, 5m)_V$

(c) $w = \cos\left(\frac{60}{2}\right) = 0.866$

$x = 0$

$y = 0$

$z = \sin\left(\frac{60}{2}\right) = 0.5$

$q = 0.866\, w + 0.5\, z$

5. (a) wheel's radius $r_w = 0.1\ m$, distance btwn wheel $w = 0.5m$

$$v = \frac{r_w}{2}(\omega_r + \omega_l) \qquad \dot{\psi} = \frac{r_w}{w}(\omega_r - \omega_l)$$

$$= \frac{0.1}{2}(0 + 4) \qquad = \frac{0.1}{0.5}(0 - 4)$$

forward speed $= 0.2\ m/s$ ∥ Yaw rate $= -0.8\ rad/s$ ∥

(b) geometry_msgs / Twist will be used to report velocity. In twist, linear.x represents forward speed, and angular.z represents yaw rate. Other elements will be not used.

6. To convert from geodetic GPS coordinates to East - North Up, (ENU)

geodetic first converted to Earth Centered - Earth Fixed (ECEF)

Step 0 Convert latitude, longitude to decimal degree representation.

Step 1 plug in latitude (ϕ) longitude (λ), altitude (h) into following equations to convert to ECEF.

$$X = [N(\phi) + h] \cos\phi \cos\lambda$$
$$Y = [N(\phi) + h] \cos\phi \sin\lambda$$
$$Z = [N(\phi)(1 - e^2) + h] \sin\phi$$

$$N(\phi) = \frac{a}{\sqrt{1 - e^2 \sin^2\phi}}$$

where $a = 6,378,137$ m

Earth's semi-major axis length

$e^2 = 6.6943799014 \times 10^{-3}$

Earth's eccentricity, squared.

Step 2 Select reference point (ϕ_r, λ_r, h_r), and convert the reference point to ECEF representation.

The reference ECEF point is shown as (X_r, Y_r, Z_r)

Step 3

Apply standard coordinate transformation to convert from ECEF to ENU.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -\sin\lambda_r & \cos\lambda_r & 0 \\ -\sin\phi_r \cos\lambda_r & -\sin\phi_r \sin\lambda_r & \cos\phi_r \\ \cos\phi_r \cos\lambda_r & \cos\phi_r \sin\lambda_r & \sin\phi_r \end{bmatrix} \begin{bmatrix} X - X_r \\ Y - Y_r \\ Z - Z_r \end{bmatrix}$$

7.

Assume we want to publish a service called mult that takes double precision floating point number as input and returns result which is (input * 9.81). The package is called example. Node is called service_example.

Step1

Create new package.

Step2

Create folder called srv in the root of the package.
Create empty folder called mult.srv (file name is defined as [servicename].srv)
Inside the mult.srv define input and output to the service as follow

```
float64 input
---
float64 result
```

```
[data type] [input name]
---
[data type] [output name]
```

All the inputs goes above – with the data type and name, and all outputs go below --- with the data type and name. This file will be used to automatically generate header file.

Step3

Perform catkin_make to generate the header file

Step4

Create C++ source file, and add following two header files.
#include <ros/ros.h>
#include <example/mult.h> (format is <[package name]/[service name].h>)

In main function, add following three lines of code
ros::init(argc, argv, "service_example");
ros::NodeHandle node;

ros::spin();

Step5

In main function, initialize the service after the NodeHandler initialization.

ros::ServiceServer srv = node.advertiseService("/mult_service", srvCallback);

The first argument is advertise name of the service, and second argument is the name of callback function when the service is requested.

Step6

Create service callback function. The callback function returns Boolean indicating result of function run. Argument passed to the functions are the request and response objects.

```
Bool srvCallback(example::mult::Request& req, example::mult::Response& res)
{
        res.result = req.input * 9.81;
        return true;
}
```

Step7

Update CMakeList.txt to define the service and include any dependency. Also, include add_exacutable line for the new node. These lines must be above catkin_package line.

Define service

```
add_service_files(
        FILES
        mult.srv
)
```

Generate message

```
generate_message(
        DEPENDENCIES
        std_msgs
)
```

Add add_dependency line to new service advertiser node

It is required to link service advertiser node with header file.

```
Add_executable(service_example srv/ service_example.cpp)
add_dependency(service_example ${PROJECT_NAME}_gencpp)
target_link_libraries(service_example ${catkin_LIBRARIES})
```

Step8

Run catkin_make to compile.

- To Run.

  rosrun example service example

- To test

  rosservice call /mult_service '{input: {data: 10.0}}'

  The result will be 98.1