AE 640A: Autonomous Navigation
Assignment #1

Due on Monday, February 5, 2018 (11:59 pm)

Problem 0: Warm up

The best way for getting started with ROS, is using the Tutorials provided on http://wiki.ros.org/ROS/Tutorials. Walk through all the Tutorials on this page, to get an overview about the ROS system. As C++ will be the language of choice throughout this course, please stick to those tutorials where needed.

Problem 1: Image Publishing & Subscription

In this problem, you need to write some simple image publishing and subscription nodes. *(50 points)*

First create a new package named **imagepub_<roll_number>**. The package would have the following dependencies: `roscpp`, `image_transport`, `cv_bridge`, `dynamic_reconfigure`. Implement the following nodes for this package:

- A node named `cb_publisher`. The node shall advertise an image topic named `cb_img`, containing a generated image of a checkerboard. Your node should have the following ROS parameters:
  - *width* - the width of the advertised image
  - *height* - the height of the advertised image
  - *square size* - the size in pixels of one square of the checkerboard
  - *frequency* - the frequency with which to publish the checkerboard image

  You need to specify default values to these parameters.

  *NOTE: For exercise you are **NOT** allowed to use any opencv functions, including the OpenCV bridge. Instead you should create and manipulate the image message directly.*
- A node named `file_publisher`, that loads an image from disk and advertises it under the topic `image`. You are allowed to use OpenCV for loading the image and the `cv_bridge` for converting an OpenCV image into a ROS image message. Your node should have the following parameters:
  - *file* - the path to the image to load
  - *frequency* - frequency used for publishing the image (use a default parameter here)

  To visualize your results you can use the `image_view` node in the `image_view` package.

You now need to create a new package named **image_enhancer_<roll_number>** for an image processing pipeline in ROS. You will implement nodes that subscribes to an image topic, performs some simple image manipulation, then publishes the resulting image on a separate image topic. Implement the following nodes for this package:

- A node named `image_changer`. The new node should have the following ROS parameters:
  - *brightness* - an integer value greater than or equal to zero
  - *contrast* - a floating point value greater than zero

After subscribing to an image topic, you need to perform brightness and contrast adjustment on incoming image. This is accomplished using a simple linear operation on the raw image data according to the formula:

$$I(x,y)' = c \times I(x,y) + b \tag{1}$$

I(x, y) 0 = c I(x, y) + b where b is the brightness change, c the contrast scaling factor, and $I(x,y)$ the image intensity value at position x, y. For multichannel images, i.e RGB, this operation can be performed on each channel individually.

*NOTE: For exercise you are **NOT** allowed to use any opencv functions, including the OpenCV bridge. Instead you should create and manipulate the image message directly.*

To visualize your results you can use the `image_view` node in the `image_view` package.

- A node named `image_dynamic_changer`. This node is similar to the `image_changer` node. However, you are going to make the node respond to parameter changes at runtime using the `dynamic_reconfigure` infrastructure.
  - Create a new directory named **cfg** in the **image_enhancer_<roll_number>** package. Create a new dynamic configuration file in that directory named `ImageChangerConfig.cfg` and make this file executable. The new configuration file should contain the appropriate entries for the `image_dynamic_changer` node.
  - Make the appropriate changes to the `CMakeLists.txt` file for dynamic reconfigure

  Test your implementation using the `reconfigure_gui` in the **dynamic reconfigure** package.

Problem 2: Attitude Tracking using Quaternion

In this problem, you need to write the MATLAB code for attitude tracking using quaternion. You can use the code developed in the class for this question. [**Option**: The transformation between quaternion and Euler angles can be used in the state function.] *(50 points)*

NOTE: The following information may proved to be useful.

The attitude dynamics of a rigid body is given by:

$$\dot{\mathbf{q}}_c = \frac{1}{2}\mathbf{q}_c \circ \mathbf{w} \tag{2}$$

$$\mathbf{J}\dot{\omega} + \omega \times \mathbf{J}\omega = \mathbf{M}_b \tag{3}$$

where $\mathbf{w} \in \mathbb{R}^4$ is the angular velocity quaternion defined as $\mathbf{w} = [0\ \omega]^T$. Here, $\omega \in \mathbb{R}^3$ is the angular velocity vector of the vehicle expressed in the body frame of reference. $\mathbf{J}$ is the inertia matrix in the body fixed frame and $\mathbf{M}_b \in \mathbb{R}^3$ is the control input for the rotation dynamics, also defined with respect to the body fixed frame. The relative orientation of the body fixed frame with respect to the inertial frame of reference has to be controlled to enable stable flight of the vehicle.

Moreover, the attitude quaternion error is defined as:

$$\mathbf{q}_e = \mathbf{q}_d^* \circ \mathbf{q}_c \tag{4}$$

where $\mathbf{q}^*$ denotes the conjugate or inverse of a unit quaternion and $\mathbf{q}_d$ is the desired quaternion. The error quaternion computed in (4) represents the proportional attitude error. The desired angular velocity vector, denoted by ω_d, is defined in the desired body frame orientation with respect to the inertial frame. Hence, ω

and ω_d are defined in different frames of reference with $\mathbf{q}_e$ as the error quaternion between these two frames of reference. Therefore, the angular velocity error is defined as:

$$\omega_e = \omega - R(\mathbf{q}_e)\omega_d \tag{5}$$

Using the above information, a PD-type controller can be written as:

$$\mathbf{M}_b = -q_{0e}\mathbf{K_q}\begin{bmatrix} q_{1e} \\ q_{2e} \\ q_{3e} \end{bmatrix} - \mathbf{K_w}\omega_e \tag{6}$$

where $\mathbf{K_q}$, $\mathbf{K_w} \in \mathbb{R}^{3\times3}$ are gain matrices. These are chosen to be diagonal matrices for convenience.

Problem 3: Path following using a PD controller

Consider the two-dimensional simple point mass dynamics of a UAV as follows:

$$\dot{x} = v\cos\theta \tag{7}$$

$$\dot{y} = v\sin\theta \tag{8}$$

$$\dot{\psi} = \frac{u}{v} \tag{9}$$

where v is the speed and u is the acceleration produced in along the direction perpendicular to the drone's velocity. You need to develop a MATLAB code to follow a straight-line trajectory using a proportional and derivative (PD) controller. You may choose alternative approaches as well. *(25 points)*