MEAM 620 Project 1 Phase 4

Lab sessions until: Tuesday, Feb 24, 2015

Report due: Mon, March 16, 2015

For this assignment, you will work as a team to implement the controller and trajectory generator you developed in the previous phases on a KMel Nano+ quadrotor platform using software developed by KMel Robotics [1].

You will need to sign up for lab time slots online, the instructions for which would be posted on Piazza. Due to limited space in the lab, only one group at a time will be allowed to use the equipment for testing under the supervision of a TA.

This document details instructions for using the hardware and software as well as the assignment and submission procedures.

1 Some Notes about the Software

1.1 Sequence Messages

The high level commands are sent to the control loop process using "sequence messages". A sequence message is a data structure with any number of fields that is sent and read via IPC, interprocess communication. You do not need to know anything about IPC to use the software. The most convenient way to pack a sequence message is to write a script which fills in the fields and then run the script when you wish to initialize the command. You may add or remove fields as you please. An example sequence message, `send_seqmsgIdle.m`, is shown below with commentary.

```
1  %Copyright KMel Robotics 2012. Must read KMEL_LICENSE.pdf for terms and conditions before use.
2
3  %This is a script that initializes a blank sequence message, adding some
4  %paths and performing some IPC functions. Use this at the top of all of
5  %your scripts.
6  init_seqmsg
7
8  %This loop initializes the relevant fields for each quadrotor you are
9  %flying.
10 for c=1:numquads
11
12     clear seq %clear out previous sequence sent to low level matlab
13     seq(1) = seq_default(); %intialize all fields to blank or default
14
15     %just sit and idle
16     seq(end).type = 700; %this is the TYPE that determines what happens in the low level code
17     seq(end).time = t_inf;%this tells the quad to hover until another message is received
18     seq(end).resetgains = 1;
19     seq(end).trpy = [1,0,0,0]; %100 grams
20     seq(end).drpy = [0,0,0]; %derivative of rpy
21     seq(end).onboardkp = [0,0,0]; %turn gains off for idling
22     seq(end).onboardkd = [0,0,0];
23     seq(end).zeroint = 1; %reset integral control to zero
24
25     seqM(c).seq = seq; %the variable seqM is a struct with metadata about the sequence
```

```
26      seq_cntM(c) = 1;
27  end
28
29  %This packs the message and sends it via IPC. Do not change this.
30  ipcm.type = 4;
31  ipcm.qn = 1:numquads;
32  ipcm.seq = seqM;
33  ipcm.seqcnt = ones(numquads,1);
34
35  sipcm = serialize(ipcm);
36
37  ipcAPIPublish(msg_name,sipcm);
```

1.2 Control Loop

1.2.1 So where do sequence messages come in to this?

The control step of the control loop runs the script `ControlLaw.m`. This script will check the "type" field of the sequence message and then call the appropriate functions or scripts. This script is already set up for the assigned sequence messages to call the assigned scripts as such:

Sequence Message File	Sequence Message ID	Calls
`send_seqmsgHover`	901	`student_control_hover.m`
`send_seqmsgGoToWaypt.m`	902	`student_control_waypt.m`
`send_seqmsgFollowWaypts.m`	903	`student_control_multi_waypt.m`

1.2.2 What about the other fields?

The number and type of fields sent in a sequence message will vary between the different message types. When the control loop switches to the next sequence, it will run some code once, that you can use for trajectory generation or any other purpose. An example is shown below, from the takeoff script.

```
1  if (setitM(qn)~=755) %The variable setitM(qn) tracks what type of sequence each quad is in.
2                       %If the quad switches sequences, this if statement will be active.
3      setitM(qn)=755;  %This changes setitM(qn) to the current sequence type so
4                       %that this code only runs once.
5
6      qd{qn}.startpose = qd{qn}.pos - [0,0,seqM(qn).seq(seq_cntM(qn)).distbelow]';
7
8      if ~isempty(seqM(qn).seq(seq_cntM(qn)).psi) %check for a desired yaw, set it
9          qd{qn}.euler_des(3) = seqM(qn).seq(seq_cntM(qn)).psi;
10     else
11         qd{qn}.euler_des(3) = qd{qn}.euler;
12     end
13
14     if(seqM(qn).seq(seq_cntM(qn)).resetgains==1) %check if reset gains is 1, set them
15         qd{qn}.onboardkp = seqM(qn).seq(seq_cntM(qn)).onboardkp;
16         qd{qn}.onboardkd = seqM(qn).seq(seq_cntM(qn)).onboardkd;
17     end
18 end %everything beyond this point runs every control loop iteration
```

This example uses the fields `psi`, `resetgains`, `onboardkp` and `onboardkd`. You can extrapolate from the example how to include your own fields. Please note that you should not need to change the onboard gains.

2 Startup Procedure

1. Make sure that the Vicon system is on and running.

2. Open a terminal and run ∼/`InitExp.sh`. This would start the following processes:
   i. central: This is the process which passes messages between the two Matlab instances.
   ii. killswitch: This is the process that controls the emergency kill switch. Rotate the killswitch and then push it down to make sure the start and kill functions are working.
   iii. High level Matlab: This is where you send the sequence messages (opens the Matlab interface).
   iv. Control Loop Matlab: This is where you run the control loop (runs in the terminal).
3. In the high level Matlab process, run the `SetUpMatlab.m` which sets some default paths and opens up the standard sequence messages.
4. In the control loop Matlab process, run mainQC.m. This will run the control loop for the duration of your experiment. This is also the process where all the data will be logged. Check that the control loop is receiving position data by verifying that the actual data field (`act`) has data.
5. Turn the quadrotor on. It should beep rapidly a few times to signal that the sensors are calibrating. It will not calibrate unless it is placed on a flat stationary surface.
6. Rotate the killswitch and verify that the first column of mode has changed to 0.
7. In high level Matlab, run `send_seqmsgIdle`. Check that the red 'sending' lights are lit on the kbee.
8. Verify that all switches on the radio controller are down (i.e., away from the user) and then turn on the radio controller. Wait for the quadrotor to beep loudly twice to verify that it is receiving signal.
9. Spin up the props on the quad by moving the left joystick on the radio to the bottom left corner.
10. Switch into serial mode by flipping the gear/fmode switch up to fmode. If you have set the idle message to send correctly, the quad should continue to idle. If the quad starts to beep loudly, switch out of serial mode and spin the props down by holding the left joystick in the same corner.
11. Assuming your commands are sending correctly, you are now ready to start controlling automatically! Open `send_seqmsgTakeoff.m` and make sure it is set to take off at a reasonable height (i.e., a height above the current position of the robot). Run this script to take off.
12. When you are ready to land, run `send_seqmsgLand.m`. Make sure the landing height is reasonable – it should be about 0.1m above the surface that the quad will land on. Spin the props down with the radio controller and turn the quad and controller off.

3 Assignment

Your task for this phase is to implement a controller and trajectory generation of your choosing and integrate it with the Matlab base of code from KMel. As a group, you will need to complete the scripts listed below and demonstrate that your control works to a TA. All provided scripts are located in the `/home/user/matlab/studentcode` folder of your student account on the computer in the lab. You are welcome to create additional files as needed to organize your scripts - just keep all of your work in your folder.

1. Read lab manual posted on the course website and watch the lab tutorial videos posted on Canvas. This step must be done before attending the lab session.
2. Complete the scripts `student_control_hover.m` and `send_seqmsgHover.m`. This should be a controller capable of stabilizing your quadrotor in hover and the corresponding high level message to initiate the controller. Have a TA check that they witnessed succesful hovering.

3. Complete the scripts `student_control_waypt.m` and `send_seqmsgGoToWaypoint.m`. This should be a controller capable of sending your quadrotor to a waypoint specified in the corresponding message. Your path between the waypoints for this step should be a line but the velocity profile is up to you. Have a TA check that they witnessed successful transition to a different waypoint.

4. Complete the scripts `student_control_multi_waypt.m` and `send_seqmsgFollowWaypoints.m`. This should be a controller capable of moving the quadrotor along a path defined by a series of waypoints sent in the corresponding message. Waypoints for this task will be loaded into the high level Matlab process using a .mat file. Some example files are in the folder `/home/user/matlab/test_waypoints`. Have a TA check that they witnessed successful completion of a path.

You are welcome to use the same control equations for each part, if your controllers works for all the tasks.

4 Submission

There are three deliverables for this assignment:

1. **Group Report**: Each group should submit a report approximately five pages in length (including plots - you don't need five pages of pure text) containing the following information:

   (a) The names of the group members

   (b) A description of any controller that you used, including equations, and the actual gains you used.

   (c) A description of your trajectory generation for waypoint following, including equations if relevant.

   (d) Plots of the actual vs desired position of the quadrotor for all three tasks. Make sure to label your axes and provide multiple views in order to translate the 3D information to the 2D paper.

   The report should be a pdf and should be sent to meam620@gmail.com.

2. **Individual "Report"**: Each person should send a one paragraph email to meam620@gmail.com with their name and group number and a short description of what they personally did for the group, and a short description of the contributions of their teammates.

3. **Code**: A .zip file of all code that you used for this phase. Send this with the report.

References

[1] KMel Robotics, http://kmelrobotics.com/