MEAM 620 Lab Manual

MEAM 620 Instructors

February 6, 2015

The purpose of this manual is to guide you through the lab experiments in meam620, in which you will work as a team to implement a controller and trajectory generator based on the KMel Nano+ quadrotor platform and software developed by KMel Robotics[1].

We will start by describing how the software works and how to write and send commands to the quadrotor. Next we will show you how to set up all the hardware correctly, including the VICON, the 620 computer and the Nano+ quadrotor. Finally, we will briefly discuss data logging.

Please read this manual at least once before going to the lab. If you have any questions, please send an email to meam620 at seas.upenn.edu.

1 Software

The software platform is developed by KMel Robotics. It is the property of KMel Robotics and it is strictly forbidden to share it with anyone. Understanding their code is not a major purpose of this course. However, we will explain it in detail here to help you integrate your controller with their software.The software consists of a set of Matlab scripts and MEX files which connect the Vicon measurements and Kbee radios.

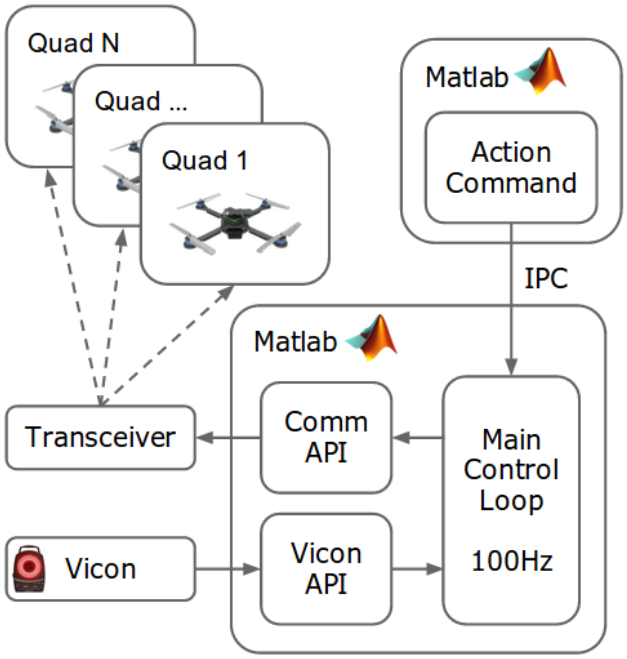

Figure 1: Communication between 2 Matlab processes

During an experiment, there will be two Matlab sessions running simultaneously. One session runs high level Matlab, which sends out sequence messages or commands. Another session is the low level Matlab (control loop Matlab), which recieves sequence messages and runs a control loop continuously. Most of your operation and programming will be done in the high level Matlab (it could also be done in the low level matlab since they share the same working directory, but it would be easier to do it in high level Matlab). The main quadrotor control script must be running in the low level Matlab during your experiment. It is also worth mentioning that all of the variables from running the main control script can **only** be accessed from the low level Matlab.

1.1 Sequence Message

The high level commands are sent to the control loop session using something called "sequence messages". A sequence message is a data structure with any number of fields that are 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 initiate 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. For now we only have one quadrotor.
10 for c=1:numquads
```

```
11      clear seq   % clear out previous sequence sent to low level matlab
12      seq(1) = seq_default(); % initialize all fields to blank or default
13
14      %just sit and idle
15      seq(end).type = 700; % TYPE that determines what happens in the low level code
16
17      seq(end).time = t_inf; % tells the quad to hover until another message is received
18      seq(end).resetgains = 1; % tells the onboard controller to prepare for a set of new gains
19      seq(end).trpy = [1,0,0,0]; % tells the quad to give a thrust of 1 gram
20      % and set desired roll, pitch and yaw to 0
21      seq(end).drpy = [0,0,0]; % tells the quad to set the derivatives of deisred rpy to 0
22      seq(end).onboardkp = [0,0,0]; % sets the p gain of the onboard controller to 0
23      seq(end).onboardkd = [0,0,0]; % sets the d gain of the onboard controller to 0
24      seq(end).zeroint = 1; % this resets integral control to 0
25
26      seqM(c).seq = seq; % the variable seqM is a struct with metadata about the sequence
27      seq_cntM(c) = 1;
28  end
29
30  % This packs the message and sends it via IPC.
31  % Do not change anything below this point
32
33  ipcm.type = 4;
34  ipcm.qn = 1:numquads;
35  ipcm.seq = seqM;
36  ipcm.seqcnt = ones(numquads,1);
37
38  sipcm = serialize(ipcm);
39
40  ipcAPIPublish(msg_name,sipcm);
```

The script might look complicated and intimidating at first sight. However, this script simply tells the quadrotor to sit there and idle with a thrust of 1 gram. The code between line 14 and line 24 are customized fields, which are a set of commands you wish to send to the quadrotor. You can define new fields inside the structure called **seq** as needed. Please note that **seq** will be cleared everytime you send a new sequence message.
Below is an exerption from a sample **send_seqmsgHover.m** which we will refer to while discussing student control code in section 1.2.

```
1   for c=1:numquads
2       clear seq % clear out previous sequence sent to low level matlab
3       seq(1) = seq_default(); % initialize all fields to blank or default
4
5       %just sit and idle
6       seq(end).type = 901; %this is the TYPE that determines what happens in the low level code
7       seq(end).time = t_inf;%this tells the quad to hover until another message is received
8
9       %PUT OTHER FIELDS HERE
10
11      seq(end).kp = [5, 5, 5]; % Set proportional gains of hover controller
12      seq(end).kd = [3, 3, 3]; % Set derivative gains of hover controller
13      seq(end).zoffset = 0.1; % Set the offset in z direction
14
15      %END OTHER FIELDS HERE
16
17      seqM(c).seq = seq; %the variable seqM is a struct with metadata about the sequence
18      seq_cntM(c) = 1;
19  end
```

1.2 Student Control Code

The control step of the control loop runs the script `ControlLaw.m`, which is called at every iteration in `mainQC.m`.

```
for j=1:num
    %update the timer
    timer(j) = GetUnixTime - time0;

    %get the positions and estimate the velocities of the vehicles
    GetQuadPositions

    %compute the control for the vehicles and send it out
    DoQuadControl

    %get ipc messages to change the sequence or kill the experiment
    GetIPCMessages

    %get feedback from the onboard computer on the vehicles
    GetQuadFeedback
end
```

In `DoQuadControl.m`, the following piece of code says run `ControlLaw.m` for each quadrotor.

```
for qn=1:nquad
    ControlLaw
end
```

Now take a closer look at `ControlLaw.m`.

```
%Copyright KMel Robotics 2012. Must read KMEL_LICENSE.pdf for terms and conditions before use.
%decide if it is time to switch sequences
endCond

%compute the control based on the current mode
if(seqM(qn).seq(seq_cntM(qn)).type==700)
    directMode
elseif(seqM(qn).seq(seq_cntM(qn)).type==55 || seqM(qn).seq(seq_cntM(qn)).type==755 || ...
    seqM(qn).seq(seq_cntM(qn)).type==7)
    setGains

    if(seqM(qn).seq(seq_cntM(qn)).type==55)
        waypointMode
    elseif(seqM(qn).seq(seq_cntM(qn)).type==755)
        takeoffMode
    elseif(seqM(qn).seq(seq_cntM(qn)).type==7)
        hoverMode
    end

    positionControl
elseif(seqM(qn).seq(seq_cntM(qn)).type==901)
    student_control_hover % Your code goes here
elseif(seqM(qn).seq(seq_cntM(qn)).type==902)
    student_control_waypt % here
elseif(seqM(qn).seq(seq_cntM(qn)).type==903)
    student_control_multi_waypt % and here
else
    hoverMode
    positionControl
end

```

```
31 %apply the safety logic
32 safetyLogic
33
34 %send out the command
35 sendCmd
```

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 (as you can see from line 20 to 25) to call the corresponding scripts.

Sequence Message File	Sequence Message ID	Student Control Code
send_seqmsgHover	901	student_control_hover
send_seqmsgGoToWaypt	902	student_control_waypt
send_seqmsgFollowWaypts	903	student_control_multi_waypt

Since you don't have write permission to the `ControlLaw.m` file, it is strongly recommended that you don't change your student control file name. Remember the `send_seqmsgHover.m` that was shown in the previous section? How do we access those customized field from our control file?

```
1  % MEAM 620 Student Hover code
2
3  if (setitM(qn)≠901) %The variable setitM(qn) tracks what type of sequence each quad is in.
4                      %If the quad switches sequences, this if statement will be active.
5      setitM(qn)=901;  %This changes setitM(qn) to the current sequence type so that this ...
           code only runs once.
6
7      %PUT ANY INITIALIZATION HERE
8      if ¬isempty(seqM(qn).seq(seq_cntM(qn)).kp) %check for a desired set of kp, set it
9          qd{qn}.kp = seqM(qn).seq(seq_cntM(qn)).kp;
10     else
11         qd{qn}.kp = [10, 10, 15]; %use the default kp
12     end
13
14     if ¬isempty(seqM(qn).seq(seq_cntM(qn)).kd) %check for a desired set of kd, set it
15         qd{qn}.kp = seqM(qn).seq(seq_cntM(qn)).kd;
16     else
17         qd{qn}.kd = [5, 5, 7]; %use the default kp
18         %these are just numbers that I made up, you will have to tune your own controller
19     end
20
21     if ¬isempty(seqM(qn).seq(seq_cntM(qn)).zoffset) %check for a desired z offset
22         qd{qn}.pos_des = qd{qn}.pos + [0; 0; seqM(qn).seq(seq_cntM(qn)).zoffset];
23         % add z offset to current position to be your new desired hover position
24     end
25     %END INTITIALIZATION
26
27 end %everything beyond this point runs every control loop iteration
28
29 %COMPUTE CONTROL HERE
30 % You shoul put your controller here, a basic example would be
31 [trpy, drpy] = controller(qd, qn, etc); % you don't need to strictly follow this
32 % But there needs something to be here that outputs trpy and drpy
```

You can see from this example that everything you have in the structure `seq` in your sequence message can be accessed in control file in the structure `seqM(qn).seq(seq_cntM(qn))`. The difference here is that `seq` is initialized in the high level Matlab and sent to the low level Matlab via IPC.

2 Hardware

The hardware that you will need to use in the lab includes the VICON motion capture system, a Mac Mini which runs Ubuntu Linux 12.04, and a Nano+ quadrotor. Our goal here is not to explain the principles behind these systems, but to show you the correct steps for setting them up.

2.1 Vicon Motion Capture System

The Vicon system is the most obvious hardware in the lab. The major components are the cameras, the controlling hardware modules, the software to analyze and present the data, and the host computer to run the software[2].
You will need to do the following steps to make sure that the Vicon system is working correctly. This is also the standard procedure to start the Vicon system in case it's not turned on when you and your team arrived at the lab.

1. Check whether the 2 stacking black computers (see figure 2), on the shelves on the wall to the left of the work station, are powered on (normally they are, with the power indicater glowing blue). If not, push the power button on each of them.

Figure 2: Vicon Hardware Module: Giganet

2. Next, open the Vicon Tracker software version 1.3.1 on the windows computer (the middle one under the stairs). The software interface is shown in figure 3. After 20 seconds, the red LEDs in front of the cameras will be on.

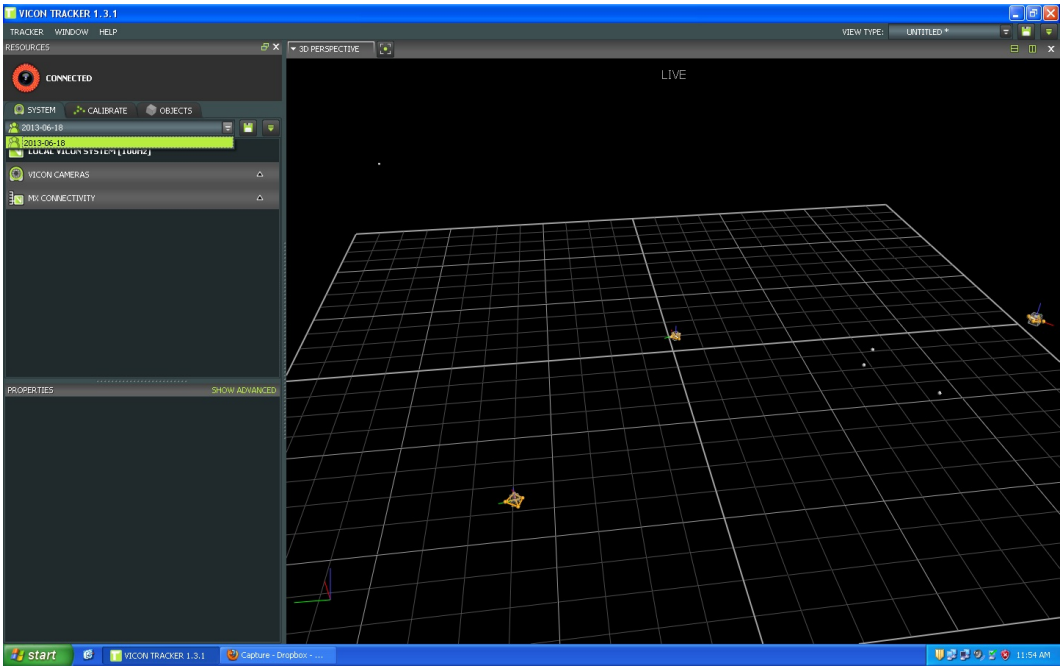

Figure 3: Vicon Tracker 1.3.1 Software Interface

3. On the left panel, make sure that the date in the listbox is the most up-to-date one, see figure 4. This means you are using the latest Vicon calibration.

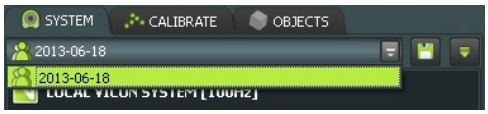

Figure 4: Vicon Tracker 1.3.1 Software System Tab

4. Go to the OBJECTS tab on the left panel (shown in figure 5, there you will see a couple of pre-defined objects that the Vicon tracker can recognize. Make sure the quadrotor you are using has the checkbox in front of it checked. For example, the one I'm using for demonstration is Nanoplus23. Each quadrotor should have a tape label with the id.

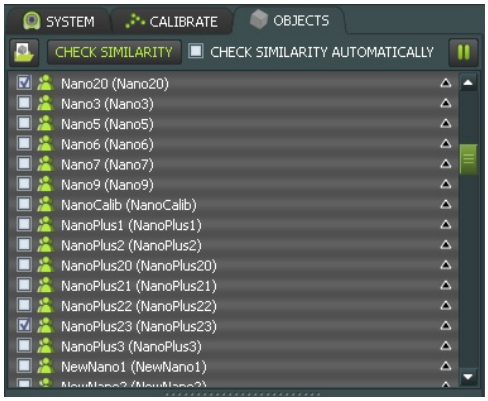

Figure 5: Vicon Tracker 1.3.1 Software Objects Tab

5. Now, if you hold that specific quadrotor and move it within the view of the cameras, your teammates will see it moving around on the right side of the interface.

6. Un-check extraneous models which are not present in the workspace. Too many similar models in view can cause mis-identification of the robot.

Up till now, you've successfully checked/set up the Vicon system and you can move on to the next step.

2.2 Nano+ Quadrotor

For lab experiments, there will probably be a Nano+ quadrotor sitting on the desk next to you.

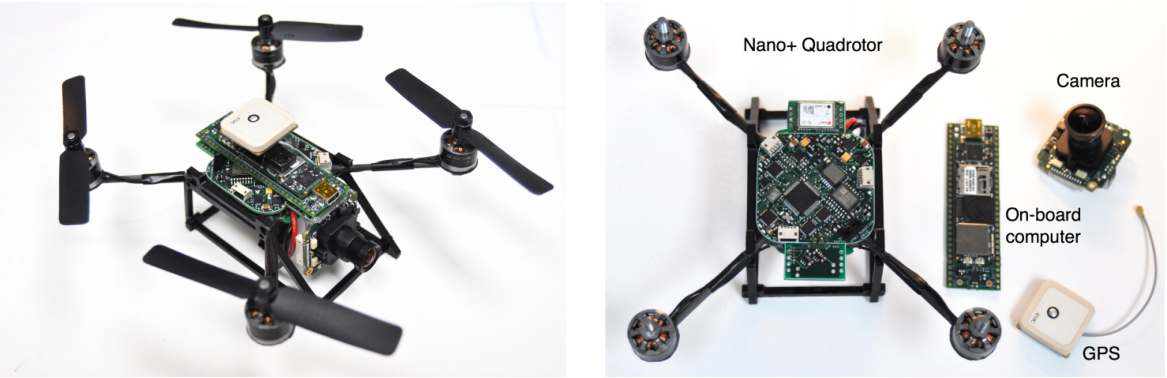

Figure 6: A quadrotor platform with VICON markers.

1. Briefly inspect the propellors to make sure they are not broken. These propellors arrive balanced and should be able to withstand crashes. Functional propellors are shown in figure 7.

Figure 7: Nano+ propellers

2. If the propellors are in good conditions, take a battery from the "Batteries Charged" bin and connect it to the quadrotor. The batteries for the Nano+ are 1350 mAh, 2 cell LiPo batteries. Connect the two blue connectors so that the red and black wires are aligned. See figure 8.

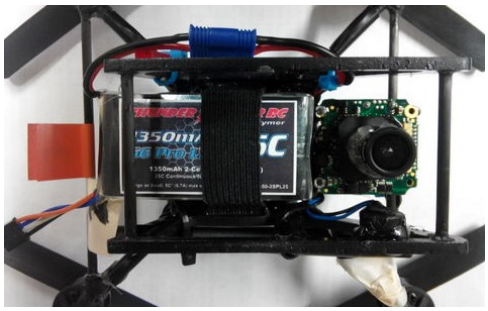

Figure 8: How to connect battery to a nanoplus quadrotor

3. Turn the quadrotor on using the switch on the top board. It should beep rapidly for a few times to signal that the sensors are calibrating. Please note that the sensor will not calibrate unless the quadrotor is placed on a flat stationary surface.

Now you have a working quadrotor that is ready to fly. Put it inside the netted flying space on top of a blue box and go to the Linux computer.

2.3 MEAM 620 Computer

Now, switch to the rightmost computer to set up the environment to run the software. It's a Mac Mini that runs Ubuntu Linux, which has a killswitch knob and a radio transmitter connected to it.
Here are the steps:

1. Before you start, make sure the VICON system is on and running correctly.

2. Make sure you are logged into the account for your group on the Mac Mini.

3. Open a terminal with 4 tabs (keyboard short-cut for creating a new tab: Ctrl + Shift + T). Start each of the following processes in one of the tabs:

   (a) central: This is the process which passes messages between the two Matlab instances. For now, in any directory, just type:
   `# central`
   into the terminal and you will see something like:
   `# Task Control Server 3.8.5 (Nov-09-09)`

`# Expecting 1 on port 1381`
`# >`
Now go to the next terminal tab.

(b) killswitch: This is the process that controls the emergency kill switch (the knob). In any directory, just type:
`# killswitch`
in the command line and the expected output would be:
`# NM IPC Version 3.8.3 (May-04-09)`
`# ... IPC Connected on port 1381`
When you rotate the killswitch, the terminal reads
`# [1373559257.281707] Published start message`
and when you push the killswitch button,
`# [1373559258.969432] Published kill message`
Rotate the killswitch and then push it down to make sure the start and kill functions are working. If your quad is about to crash, hitting the killswitch would stop the propellors immediately and thus reducing damage.

(c) High level Matlab: This is where you edit all your programs and send the sequence messages. Execute the following in any directory:
`# matlab`
and wait for Matlab to launch.

(d) Control loop Matlab: This is where you run the control loop. Execute the following in any directory:
`# matlab -nodesktop`
and the Matlab process will run in the terminal

4. **For *Project 2 Phase 3* ONLY!** After turning on the quadrotor, wait for 30 seconds for it to connect to a wifi network. Then open a new terminal tab and type in:
`# nanoplus22`
to ssh into the onboard gumstix. Then type:
`# ./startSendData`
to enable data streaming.

5. Next, you will need to check the communication between your quadrotor and Matlab. In the high level Matlab process, type in the following command:

```
edit init_exp.m
```

This will open the script `init_exp.m`, check that `usequad(1)` is set to the desired quad ID and `qd{qn}.dev` is set to `ttyUSB0`.

```
%set the ids of the vehicles to be controlled
usequad(1) = 708; % Use edit quadInfo.m to find the correct quad
usequad(2) = 0; % The rest are all zeros, since you will fly only one quadrotor
usequad(3) = 0;
usequad(4) = 0;
% some other lines of codes
for qn=1:nquad;
    qd{qn}.dev = '/dev/ttyUSB0'; %set the device
end
```

In this example, I'm using Nanoplus23, so the corresponding ID is 708. But you might not be using the same quadrotor as me, so here is how to find the correct ID.

(a) Identify the number tag on the quadrotor, for example, 23 (note that this is different from the ID we mentioned above).

(b) In your high level Matlab process, run

```
edit quadInfo.m
```

This will open another script that records all quadrotors and its corresponding ID and configuration.

(c) Try to find the case which reads

```
case 708
    qd{c}.type = 7;
    qd{c}.name = 'NanoPlus23';
    qd{c}.calib = strcat(qd{c}.name,'Calib.mat');
    qd{c}.kbeechannel = 0;
    qd{c}.kbeeid = 3;
    qd{c}.driver = @kQuadInterfaceAPI;
```

So you will know that the `usequad` field in `init_exp.m` for Nanoplus23 should be `708`.

Till this point, you have successfully configured the command environment for your lab.

2.4 Startup Procedure

Now is the time to fly the quadrotor using your software. Simply follow these steps:

1. First of all, wear protecting glasses.
2. If you are following this manual, you quadrotor is already turned on and sitting in the flying space in front of you, with the net down. If this is not the case, find someone to help you lower the net.
3. 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. The expected output would be something like this:
   `# qdi seq:j des:x, y, z act:x, y, z modes:a,b`
   If the second number after mode (`a`) is not 0, you need to restart `mainQC.m`.
4. Without doing anything else, just run each of your sequence messages to see whether they can run correctly. If none of them crashes, stop and re-run `mainQC.m`.
5. Turn the killswitch about the z axis and verify that the first column of mode (`a`) has changed to 0.
6. In high level matlab, run `send_seqmsgIdle.m`. Check that the red 'sending' lights are lit on the kbee.
7. Verify that all switches on the radio controller are down (i.e., away from the user). Turn on the manual radio controller. Wait for the quadrotor to beep loudly twice to verify that it is receiving signal.
8. Spin up the propellors on the quad by holding the left joystick on the radio to the bottom left corner.
9. 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. See figure 9.

Figure 9: Instruction on how to operate the radio

10. Assuming your commands are sending correctly, you are now ready to start controlling automatically! Open `send_seqmsgTakeoffAll.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.

11. From now on, you can send your own sequence messages, (e.g., `send_seqmsgHover.m`) to see how well your controller is doing.

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.

3 Some Notes on the Software

3.1 Important Variables

Now that you've understood the basic structure of KMel's software platform. Here are some notes that might be helpful to you for developing your controller.

1. The `qd` cell.

```
for c=1:nquad
    qd{c}.phi = 0;
    qd{c}.theta = 0;
    qd{c}.psi = 0;

    qd{c}.euler = zeros(3,1);
    qd{c}.pos = zeros(3,1);
    qd{c}.pos_raw = zeros(3,1);
    qd{c}.vel = zeros(3,1);
    qd{c}.vel_body = zeros(3,1); % Linear velocity in the body frame
    qd{c}.omega = zeros(3,1); % Angular velocity in the body frame
    qd{c}.Rot_last = eye(3);
    qd{c}.framelast = 0;
    qd{c}.viconid = [];

    qd{c}.euler_des = zeros(3,1);
    qd{c}.pos_des = zeros(3,1);
    qd{c}.vel_des = zeros(3,1);
```

```
19
20      %integral terms
21      qd{c}.phi_int = 0;
22      qd{c}.theta_int = 0;
23      qd{c}.th_int = 0;
24      qd{c}.yaw_int = 0;
25
26      %onboard attitude gains
27      qd{c}.onboardkp = [170,170,170];
28      qd{c}.onboardkd = [17,17,17];
29
30      qd{c}.vtime = -inf; %the last time a good vicon value was received
31
32      qd{c}.safetymode = 0;    %0 is normal, 1 is safety descent, 2 is killed
33      qd{c}.yawcmdsafety = 0;  %yaw angle to which quad is commanded during
34                               %safety descent mode, set in safetyLogic.m
35
36      %     qd{c}.stopangletime = 0;
37  end
```

`qd` is probably the most important variable that you will use in your controller. The structure of `qd` is pre-defined by the script `init_variables.m`. The current states of the quadrotor is located in qd, which is a cell array of stucts. Each cell contains one struct which contains many fields, the most relevant of which are detailed below:

Field	Information
`pos`	Current position of quadrotor, with slight filtering
`pos_raw`	Current position of quadrotor, raw measurement from Vicon
`vel`	Current velocity of quadrotor (estimate from filtered vicon data)
`euler`	Current euler angles of quadrotor
`pos_des`	Desired position of quadrotor
`vel_des`	Desired velocity of quadrotor
`euler_des`	Desired euler angle of quadrotor

It is recommended that you only change the following three fields, and remember that they are all 3-by-1 column vectors:

```
1  qd{c}.euler_des = zeros(3,1);
2  qd{c}.pos_des = zeros(3,1);
3  qd{c}.vel_des = zeros(3,1);
```

The reason why you should use these fields is for debugging purposes.

```
1  for qn=1:nquad
2          if(printcnt==printinterval)
3              display(sprintf('qd%d seq:%d des:%4.3f, %4.3f, %4.3f act:%4.3f, %4.3f, ...
                   %4.3f modes:%d,%d',...
4                  qn,seq_cntM(qn),qd{qn}.pos_des(1),...
5                  qd{qn}.pos_des(2),qd{qn}.pos_des(3),qd{qn}.pos(1),...
6                  qd{qn}.pos(2),qd{qn}.pos(3),killflag,qd{qn}.safetymode));
7          end
8      end
9  end
```

The `DoQuadControl.m` will print out the desired and current position of the quad to the low level Matlab using `qd{qn}.pos_des` and `qd{qn}.pos`. So if you use `qd{qn}.pos_des` in your controller, it will be automatically displayed on the screen.

3.2 Sending Commands

Unlike in simulation, the quadrotor you will be using has separate control loops for position and attitude. The attitude control loop runs onboard the robot and you will not have any power to change this at all. Instead, you will command the desired roll, pitch, and yaw angles and assume that the attitude controller can reach them. You will also command the desired total thrust, or force, which is a scalar along the z axis of the quadrotor. The command looks like:

```
1 trpy = [thrust roll pitch yaw];
```

You can set this anywhere in your controller. This will be sent right after the controller script you write executes. One thing to notice is that `thrust` is in gram, so what ever force `F` you have from your controller, you will need to do the following conversion before send it to the quadrotor.

```
1 thrust = F / gravity * 1000; %convert force in newton to thrust in gram
```

Remember, `trpy` is just one variable in the base workspace. Don't wrap `trpy` into `qd{qn}` or other structures, otherwise your quad won't fly.

3.3 Logging Data

Most relevant data is currently logged by the control loop every iteration (100 Hz). The variables where the most used data is stored are shown below. All data is stored as $dim \times num \times nquad$ matrices, where dim is the dimension of data of interest, num is a finite number of control loop iterations (default set to 250000) and nquad is the number of quadrotors being controlled (default 1).

Variable	Structure	Information
`ViconData`	`[roll; pitch; yaw; x; y; z]`	Logged position and orientation
`VelSave`	`[xd; yd; zd]`	Logged estimated velocity of quadrotor
`trpySave`	`[thrust; roll; pitch; yaw;]`	Logged control commands
`DesPosSave`	`[x; y; z]`	Logged desired position

If you are curious about what other data is logged, you can type whos into your control loop matlab process (after stopping the loop) to see all of the variables in the workspace. If you wish to save your experimental results, you will need to save it from the control loop process. You can save your whole workspace by typing

```
1 save('my_file_name')
```

References

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

[2] VICON MX. http://www.vicon.com/System/TSeries.