Laboratory D: Velocity Control & Navigation

Assistants: Nicolas Dousse and Julien Lecoeur

28.10.2014

INTRODUCTION

In the previous TP, we modeled and tuned a PID controller that allows to control the attitude of the LE quad. The aim of this TP is to implement a new layer of automation on the autopilot. A velocity controller will be implemented where the user can control the LE quad by giving velocity inputs. This controller will also be used to perform automatic GPS navigation.

PREPARATION

1. Open the Lab_D folder, double click on the Mavric_MobRob Atmel Studio solution file,
2. Compile the code (shortcut: F7),
3. Connect the MAV'RIC board to your computer with the USB cable,
4. Set the autopilot in programming mode (i.e. press the reset button while pressing the programming button) and double click on "dfu-programming.bat" to flash the autopilot.
5. Make sure your XBee modules are paired:
   a) Plug one XBee module on the MAV'RIC board and the other one on the ground station board,
   b) Connect the ground station to your computer with the USB cable,
   c) Open QGroundControl, select the good COM port on the top right corner of the window and click "Connect".

If the connection works you can go on, if it does not, redo the steps from Lab 1 to pair the two XBee modules.

POSITION ESTIMATION DRIFT

In the previous TP, an attitude controller was designed in order to control the LE quad. The goal of this TP is to add an automation layer on the top in order to achieve higher level of autonomy.
With the sensors available on the autopilot, it is possible to obtain an estimation of the velocity and of the position by simple/double integration of the acceleration:

$$\mathbf{v}(t) = \int \mathbf{a}(t)\mathrm{d}t$$
$$\mathbf{x}(t) = \int\int \mathbf{a}(t)\mathrm{d}t^2$$

and in the discrete time domain:

$$\mathbf{v}(t+1) = \mathbf{v}(t) + \mathbf{a}(t)\cdot dt$$
$$\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{v}(t+1)\cdot dt$$

The acceleration measurement coming from the accelerometer can be modeled as

$$\mathbf{a}(t) = \mathbf{a_{acc}}(t) + \chi(t) + \varepsilon \tag{0.1}$$

where $\chi(t)$ is a time varying noise and ε is a constant noise.
In this TP, the autopilot is set in real mode as start mode. What happen to the position estimation? Why?
Answer: _____

_____

Therefore, an extra sensor is required to give an absolute position estimation such as GPS, a barometer or a VICON system (i.e. a system of cameras placed in a room giving a precise position at very high frequency). The error of the position/velocity estimation can be corrected by one of these sensors.
By mixing the fast update rate from the accelerometers and the (more) precise estimation of GPS, we can achieve high performances.
In simulation mode, the signal of the GPS is simulated. The code for the position estimation is given in the position_estimation.c file. The function position_estimation_update is called at each time step and performs two actions:

- First, it calls the position_estimation_integrate,
- Second, it calls the position_estimation_correction function.

What is performed by these functions?
Answer: _____

_____

Now that we have position and velocity estimations that do not drift, we can add an new automation layer.
Before going further, switch your board to simulation mode

1. Go to src/central_data.c file,
2. Comment line 121, where the simulation mode is set to HIL_OFF,
3. Uncomment line 122, where the simulation mode is set to HIL_ON,
4. Compile the code (shortcut: F7),
5. Set the autopilot in programming mode (i.e. press the reset button while pressing the programming button) and double click on "dfu-programming.bat" to flash the autopilot.

Now your board is again in simulation mode. It means that the sensors (accelerometer, gyroscope, magnetometer, barometer and GPS) are simulated and that GPS and barometer are used to correct the position estimation. You can observe that the quadrotor is not drifting anymore.

EXERCISE 1: VELOCITY CONTROLLER

In this exercise, the goal is to implement a velocity controller. Instead of an attitude command, the joystick will provide a velocity command. The velocity controller will be implemented as a new layer in the cascade controller introduced in Lab C.
The PID gains for the rate and attitude controllers are given in the conf_stabilisation_copter.h file. All gains for the velocity controller are currently set to zero.

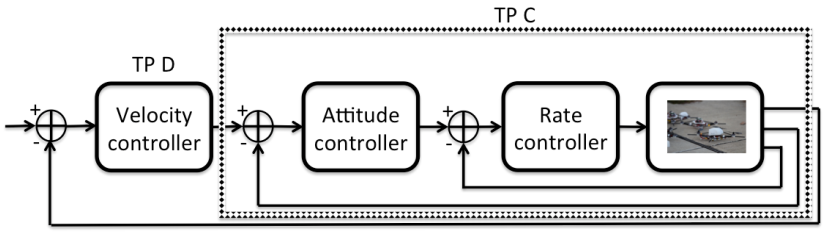

Figure 0.1: The cascade controller, the rate and attitude controller were designed in TP C. This TP will be about the design of a velocity controller.

MAPPING THE JOYSTICK INPUTS TO A VELOCITY VECTOR

The first step is to map the joystick inputs to a velocity vector, named tvel.

$$v_x = \text{“sign”} \cdot 10 \cdot \text{“joystick\_parsing->controls->XXX”} \cdot \text{MAX\_JOYSTICK\_RANGE}$$
$$v_y = \text{“sign”} \cdot 10 \cdot \text{“joystick\_parsing->controls->XXX”} \cdot \text{MAX\_JOYSTICK\_RANGE}$$
$$v_z = \text{“sign”} \cdot 1.5 \cdot \text{“joystick\_parsing->controls->XXX”} \cdot \text{MAX\_JOYSTICK\_RANGE}$$
$$\text{yaw} = \text{“joystick\_parsing->controls->XXX”} \cdot \text{MAX\_JOYSTICK\_RANGE}$$

1. Open the Library/control/joystick_parsing.c file,
2. Look for the joystick_parsing_get_velocity_vector_from_joystick (line 176),
3. The vector tvel is the velocity command vector. In order to map the value of the tvel vector by the corresponding axes of the joystick, replace the “XXX” by the corresponding axes of the joystick:
   a) Pitch value: joystick_parsing->controls->rpy[PITCH],
   b) Roll value: joystick_parsing->controls->rpy[ROLL],
   c) Yaw value: joystick_parsing->controls->rpy[YAW], and
   d) Trust value: joystick_parsing->controls->thrust.
4. Replace the sign of the inputs by 1.0 or -1.0 in the formula to map the reference frame axes with the joystick axes:

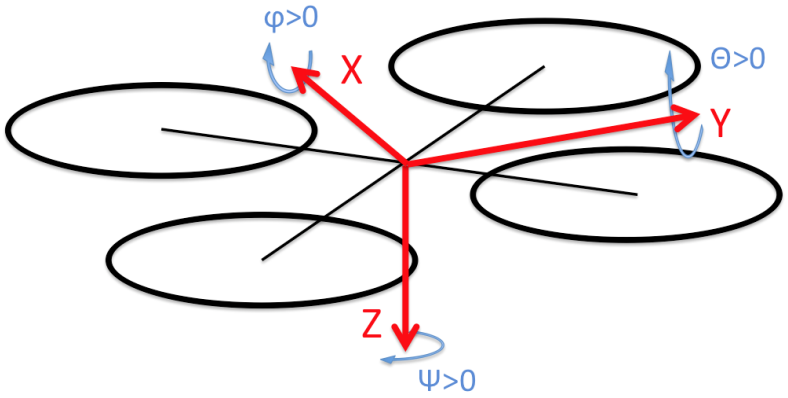

Figure 0.2: The NED reference frame and the positive sens of angular rotations.

   a) The velocity vector is in the NED reference frame (for North East Down):
      i. Positive X axis is front,
      ii. Positive Y is right, and
      iii. Positive Z is down.
   b) The joystick signs are:

i. Positive when pitching up, i.e. pulling on the stick,

ii. Positive when rolling right, i.e. pusing the stick on the right,

iii. Positive when (yaw) turning right, i.e. rotating the joystick to the right along a vertical axis, and

iv. Thrust is increasing when quad going up.

5. The joystick values are normalized between -1 and 1. Therefore, maximal deflections of the joystick would provide a maximal velocity of $1 m/s$ on each axis. In order to have a more reactive quad, scale the velocity vector as shown in the formula above.

IMPLEMENTATION OF THE VELOCITY CONTROLLER

The next step is to implement the velocity controller.

1. Open the Library/control/stabilisation_copter.c file,

2. Look for the stabilisation_copter_cascade_stabilise function (line 82),

3. The input to a PID controller is the error between the command and the measured value. The rpyt_errors vector is the error vector that is used to perform the PID controller, replace the $0.0f$ (lines 121-124) by the difference between the velocity command vector and the velocity estimation:

   a) As command, use: input.tvel[X, Y, Z],

   b) As feedback values, use the velocity estimation expressed in the bodyframe:

   stabilisation_copter->pos_est->vel_bf[X, Y, Z],

   c) Map the axis of the error vector with the desired attitude input (think about the sign), i.e. from X,Y and Z and YAW to ROLL, PITCH, YAW and THRUST,

   d) Note that there is no velocity control on the yaw axis (only attitude and rate controllers), therefore, the input is directly equal to the output: rpyt_errors[YAW] = input.rpy[YAW].

4. Uncomment line 131, where the stabilisation_run function is called. This is where the PID control is performed.

   If we tried to tune the gain with the actual controller, an integral part would be necessary on the vertical axis. Why? What would happen if we omit the integral part?

   Answer: _____

   _____

   a) Add the constant THRUST_HOVER_POINT to compensate gravity (line 133).

5. Uncomment the line where the theading of the output is set (line 134). This line is used to feed forward the yaw command to the attitude controller,

6. Uncomment the line where the output of the stabiliser is mapped with the input structure (line 135). This will allow the next step of the cascade controller (i.e. the attitude controller) to be run afterwards.

Load your new code on the autopilot.

1. Compile the code (shortcut: F7).
2. Set the autopilot in programming mode (i.e. press the reset button while pressing the programming button) and double click on "dfu-programming.bat" to flash the autopilot.
3. Open QGroundControl, select the good COM port on the top right corner of the window and click "Connect".
4. Arm the autopilot (button 1 of the joystick), don't forget that you should be in attitude control mode (button 3 of the joystick) in order to arm the MAV,
5. Switch to Velocity control mode (button 5 of the joystick),
6. Fly around.

The velocity command from the joystick can be applied in the local or in the global reference frame. Until now, the velocity command was applied in the local frame. During a windy day, what is the main drawback of giving the velocity command in the local frame? What could be a possible solution to overcome this issue?
Answer: _____

_____

OPTIONAL: PERFORMING THE PID IN THE GLOBAL FRAME

The error for the PID control can be computed in the global frame.

1. Uncomment the top part of the velocity controller (line 100-117),
2. Uncomment the variable definition above the switch statement (line 89-91),
3. Map the axis of the error vector with the desired velocity input (think about the sign), i.e. from X,Y and Z and YAW to X, Y, Z and THRUST:
   a) As command use: input.tvel[X, Y, Z],
   b) As feedback values, use the velocity estimation expressed in the global frame: stabilisation_copter->pos_est->vel[X, Y, Z].
4. Again, there is no velocity control on the yaw axis (only attitude and rate controllers), therefore, the input is directly equal to the output: rpyt_errors[YAW] = input.rpy[YAW],
5. Instead of adding, substract THRUST_HOVER_POINT to the thrust output in order to compensate gravity (remember that a positive Z velocity is pointing downwards),

6. As the attitude controller is given in the local reference frame, we have to map back the velocity PID controller output into the local frame. The rpy_local structure gives the output value of the controller in the local frame:
    a) Uncomment lines 143-149.
    b) Map input.rpy[ROLL] with the corresponding component of the rpy_local.v vector (think about the sign),
    c) Map input.rpy[PITCH] with the corresponding component of the rpy_local.v vector (think about the sign),
    d) Map input.thrust value with the value of the velocity_stabiliser.output.thrust of the controller (think about the sign).

Load your new code on the autopilot.

1. Compile the code (shortcut: F7),
2. Set the autopilot in programming mode (i.e. press the reset button while pressing the programming button) and double click on "dfu-programming.bat" to flash the autopilot,

Test your new code.

1. Open QGroundControl, select the good COM port on the top right corner of the window and click "Connect",
2. Arm the autopilot (button 1 of the joystick), don't forget that you should be in attitude control mode (button 3 of the joystick) in order to arm the MAV,
3. Switch to Velocity control mode (button 5 of the joystick),
4. Fly around.

EXERCISE 2: "DIRECT TO" NAVIGATION STRATEGY

The goal of this exercise is to implement a "Direct to" navigation strategy. This strategy is a simple navigation strategy that takes the direction to the goal waypoint (point B on Fig.0.3) from the current position (point A on Fig.0.3) of the robot and returns a velocity command in this direction.

1. Open the Library/control/navigation.c file,
2. Look for the navigation_set_speed_command function (line 117),
3. The dir_desired_bf gives the vector towards the goal waypoint in the local frame,
4. Implement and test one after the other the following function as goal speed for the navigation;

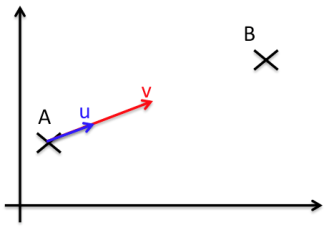

Figure 0.3: The principle of “Direct to” navigation.

a) A constant speed:

$$desired\_speed = cruise\_speed;$$

b) A speed proportional to the distance to the goal:

$$desired\_speed = dist2vel\_gain * norm\_AB;$$

c) A speed proportional to the distance to the goal and bounded by the cruise speed (similar to Fig.0.4)(Useful function maths_f_min(a,b)):

$$desired\_speed = ???$$

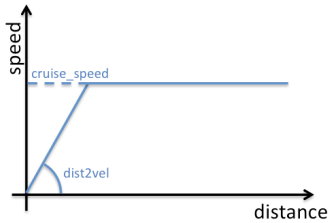

Figure 0.4: The relationship between the speed and the distance to the goal.

5. Compute the normalized vector **u** such that

$$\mathbf{u} /\!/ \mathbf{AB}$$
$$\|\mathbf{u}\| = 1$$

6. Replace the $0.0f$ such that
   - tvel is pointing towards the goal and
   - has a norm of $desired\_speed$.
7. [Optionnal] Once you tested the three solutions:
   a) Bound the desired velocity such that the vertical component of this velocity is below navigation->max_climb_speed (see Fig.0.5),

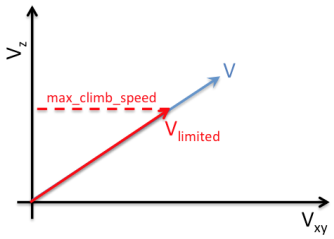

Figure 0.5: The bounds on the vertical component of the velocity.

Load your new code on the autopilot.

1. Compile the code (shortcut: F7).
2. Set the autopilot in programming mode (i.e. press the reset button while pressing the programming button) and double click on “dfu-programming.bat” to flash the autopilot.

Test your new code.

1. Set waypoints to follow,
   a) Open QGroundControl, select the good COM port on the top right corner of the window and click “Connect”,
   b) Go to the Mission tab,
   c) Zoom on the map to the location of the MAV,
   d) On the bottom of the tab, there is the Mission Plan widget, if not, click on “Tool Widget”→“Mission Plan”,
   e) Click on “Edit Waypoints”,
   f) Double-click on the position of the map where you want to set a waypoint,
   g) Repeat in order to have 3-4 waypoints,
   h) Click on “Set”.
2. Arm the autopilot (button 1 of the joystick), don’t forget that you should be in attitude control mode (button 3 of the joystick) in order to arm the MAV,
3. Switch to GPS navigation control mode (button 6 of the joystick),
4. If the MAV hasn’t taken off yet, increase the throttle above a quarter of its range,
5. See the MAV fly around folowing the waypoints.