1 MPU Code

1.1 ROS Communicate

Copy the contents in the folder ROS to the folder Core of the STM32 project. Change the including headers and uart prot in STM32Hardware.h to adapter the settings of the project. The core code can be done in the file mainpp.cpp and put the functions setup() and loop() defined in this file to the correct place of main.c.

Note don't set dim of the MultiArray message. Just use the following code:

1.2 Manual(RC Transimitter) input switch

The PWM signals from receiver are forward to STM32. In STM32 we use "PWM INPUT" functions of the timer to capture the frequence and the duty cycle of the pwm signals. Channel 1 of TIM5 is used to capture servo input and Channel 1 of TIM15 is used to capture ESC input. The output of servo and esc signals in STM32 are Channel 1 and Channel 2 of TIM3. Note output frequences would be the same as the two output channels are in the same timer. This is feasible since the frequences of the input signals are from the same receiver.

```
std_msgs::Float32MultiArray forces;
forces.data = new std_msgs::Float32MultiArray::_data_type[8];
forces.data_length = 8;
//do not use the following:
//forces.layout.dim = new std_msgs::MultiArrayDimension();
```

1. Select "PWM input on CH1", and set PSC to (X-1) where X is the frequence of APB1 Timer Clocks and APB2 Timer Clocks in MHz (Please set the two timer clocks the same to simplify settings). This leads the capture precision be 1/1M=1us. This is reasonable since the max value of the 16-bit ARR is 65535, the lowest frequence which can be detected is 1M/65535 = 15Hz. Typically the frequence of PWM signals for RC are 50Hz, the count is around 1M/50Hz = 20000. Note the TIM3 has a 32-bit ARR. Leave other settings default except check the global interrupt.

2. Insert the callback function:

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){

    if(htim->Instance==TIM5 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1){
            uint32_t cl = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
            uint32_t ch = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);


            freq1 = (float) TIMER_CLOCK_FREQ / (cl + 1);
            duty1 = (float) 100 * ch / cl;
            __HAL_TIM_SetCounter(htim,0);
```

```
                HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_1);
                HAL_TIM_IC_Start(&htim5, TIM_CHANNEL_2);
        }else if(htim->Instance==TIM15 && htim->Channel == HAL_TIM_ACTIVE_CHANN
                uint32_t cl = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
                uint32_t ch = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);
                __HAL_TIM_SetCounter(htim,0);
                freq2 = (float) TIMER_CLOCK_FREQ / (cl + 1);
                duty2 = (float) 100 * ch / cl;
                HAL_TIM_IC_Start_IT(&htim15, TIM_CHANNEL_1);
                HAL_TIM_IC_Start(&htim15, TIM_CHANNEL_2);
        }
}
```

Of course you have to start the timer inside the setup function after the timers is initialized. TIMER_CLOCK_FREQ is the APB Timer Clock.

3. Forward the frequence and duty cycle to TIM3. Using the following formula to set the ARR of TIM3:

$$ARR = \frac{TIMER\_CLOCK\_FREQ}{frequence * (PSC + 1)} - 1 \tag{1}$$

Then using

```
__HAL_TIM_SetAutoreload()
```

to set ARR value of TIM3. Note this value only has to be set once. And the Counter values of Channel 1 and Channel 2 are set to ARR*duty1 and ARR*duty2 respectively. This can be done inside callback function HAL_TIM_IC_CaptureCallback.