Early Image Processing Test on UAV

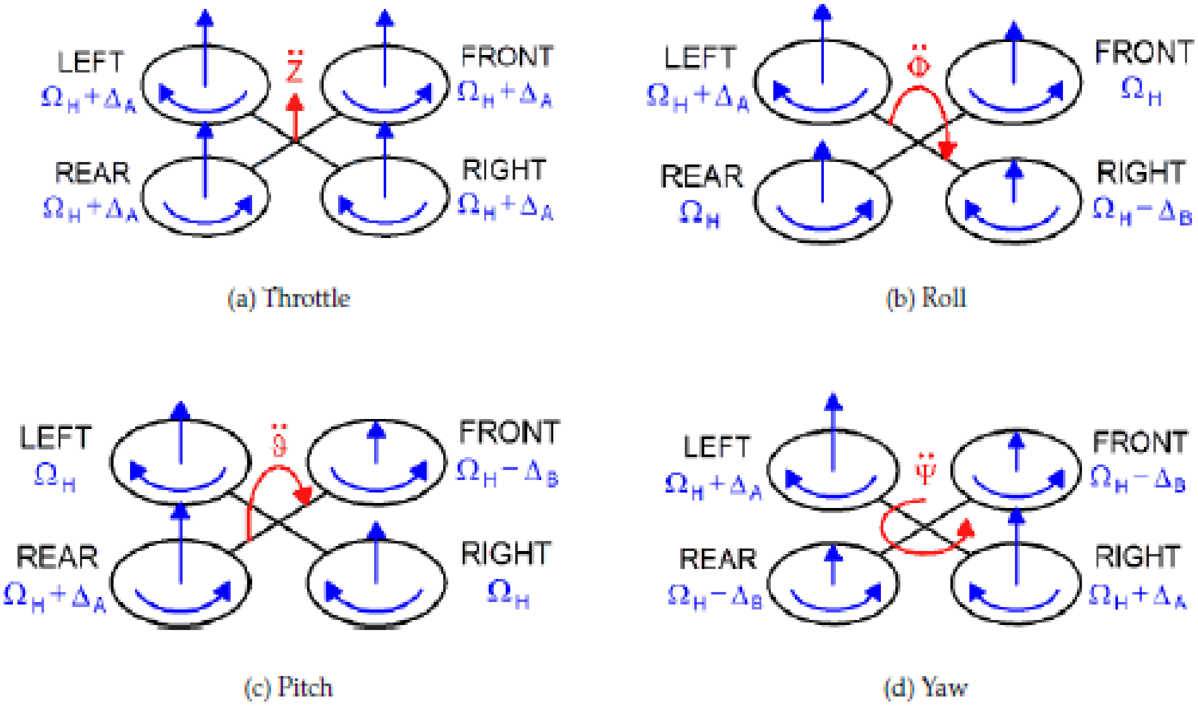
Control System

By: Pratik J Borhade, Teng Huang

Introduction: This is report on how the control system part of AR Drone parrot was implemented. In this report we will start from the basic idea, describe changes needed for improvement in control, and then how we arrived on current control system design.

AR Drone Control:

1) Drone Movements:



(a) Throttle

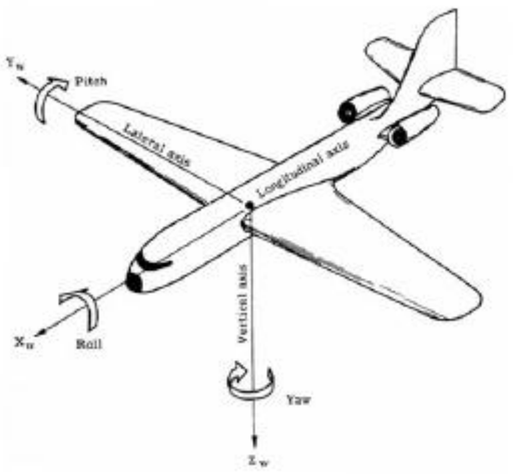
(b) Roll

(c) Pitch

(d) Yaw

Figure 2.1: Drone movements

Here Ω stands for Base speed at which blade rotates to keep drone stable and Δ is small deviation in speed.
Throttle : Makes Drone to go up and down i.e to increase or decrese it's altitude.
Pitch: Pitch make drone to tilt a little forward, which increases it forward speed.
Roll: It is same as pitch but in left - right direction.
Yaw: It makes drone to rotate around its axis.

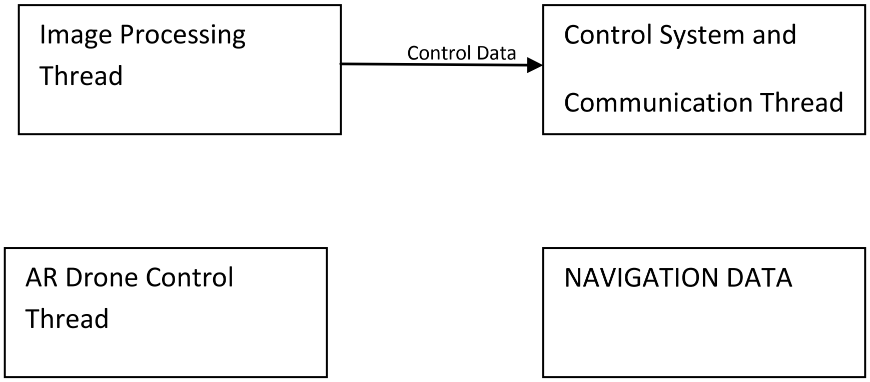


2) AR Drone SDK:
   AR Drone SDK is little different as it does not have any main function.
   `C_RESULT ardrone_tool_init_custom(int argc, char **argv)` - acts as a intitialisation function.
   Similarly `C_RESULT ardrone_tool_shutdown_custom()` – acts as a shutdown function.

   ARDrone SDK has also implemented its own threading system. For every thread we make, we have to enter the function name in Thread Table.

<u>Control System 1.0:</u>

The initial control system design was simple. We used two threads one was used for image Capturing, Image Processing, GUI and Control System. Other thread is just sending commands to AR Drone and wait for some given time. AR Drone SDK has already two threads running one of them is control thread and other is Navigation Data Update thread.



In this model Image Processing Thread did following operation:

A) Image Capture and Transfer to OpenCV Variable.

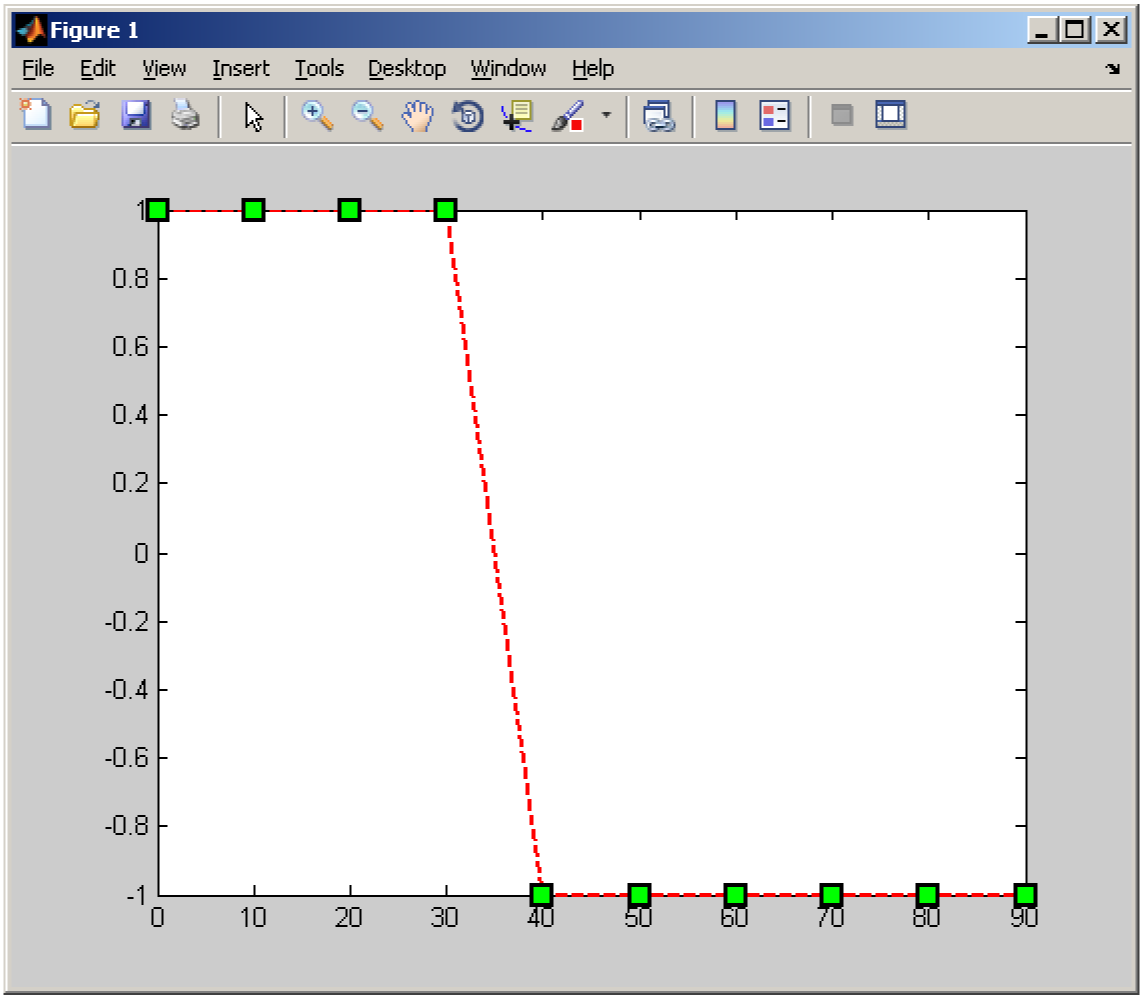
B) Show the Image in GUI.

C) Use Image Processing Algorithms on the Image.

D) Manipulate the Controls according to Output of Image processing system.

The Problem:

In this the Image Processing Block Delays are much higher than the Control System Block. These delays include delay of waiting for next frame, fixed delay of 20ms to display the image and a variable delay to process the image which depends on algorithm used. While the delay in control system block was only 10 ms constant. The following is discrete output produced in discrete form.



Time (milliseconds) vs Control Value

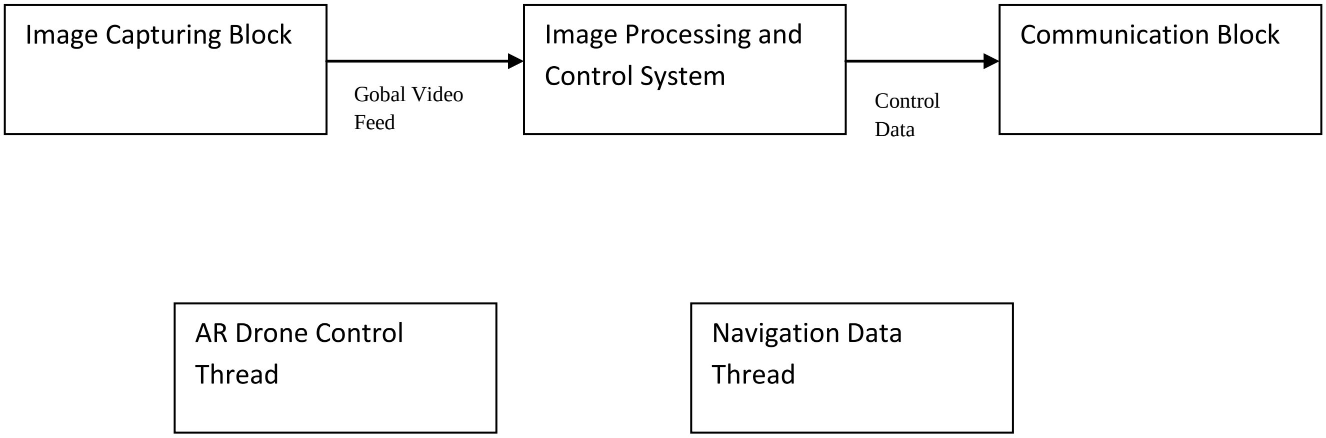
Here Red Line indicates the value obtained by Image Processing block and Green Points show the value sent by the Control System Block.

The biggest problem causing instability was if image processing block was too late to change the control or if it is crashed then control system block will continue to send the previous command which will lead to failure of system and crashing of drone.

Performance: The major fault in performance was the Image Processing Block and Image Capturing Block are on the same thread. So no new frames were captured when Image Processing Block was under Progress. This reduced the FPS of output image greatly which hampered performance of native AR Drone Library blocks like ardrone control.

Control System 2.0:

In Control System 1.0 the FPS was very low. So we decided to separate the block of Image Processing and Image Capturing. As it will unlock the Image Mutex Control sooner this way the native system won't be affected due to instability in image compression block. The following is block diagram of resultant system.

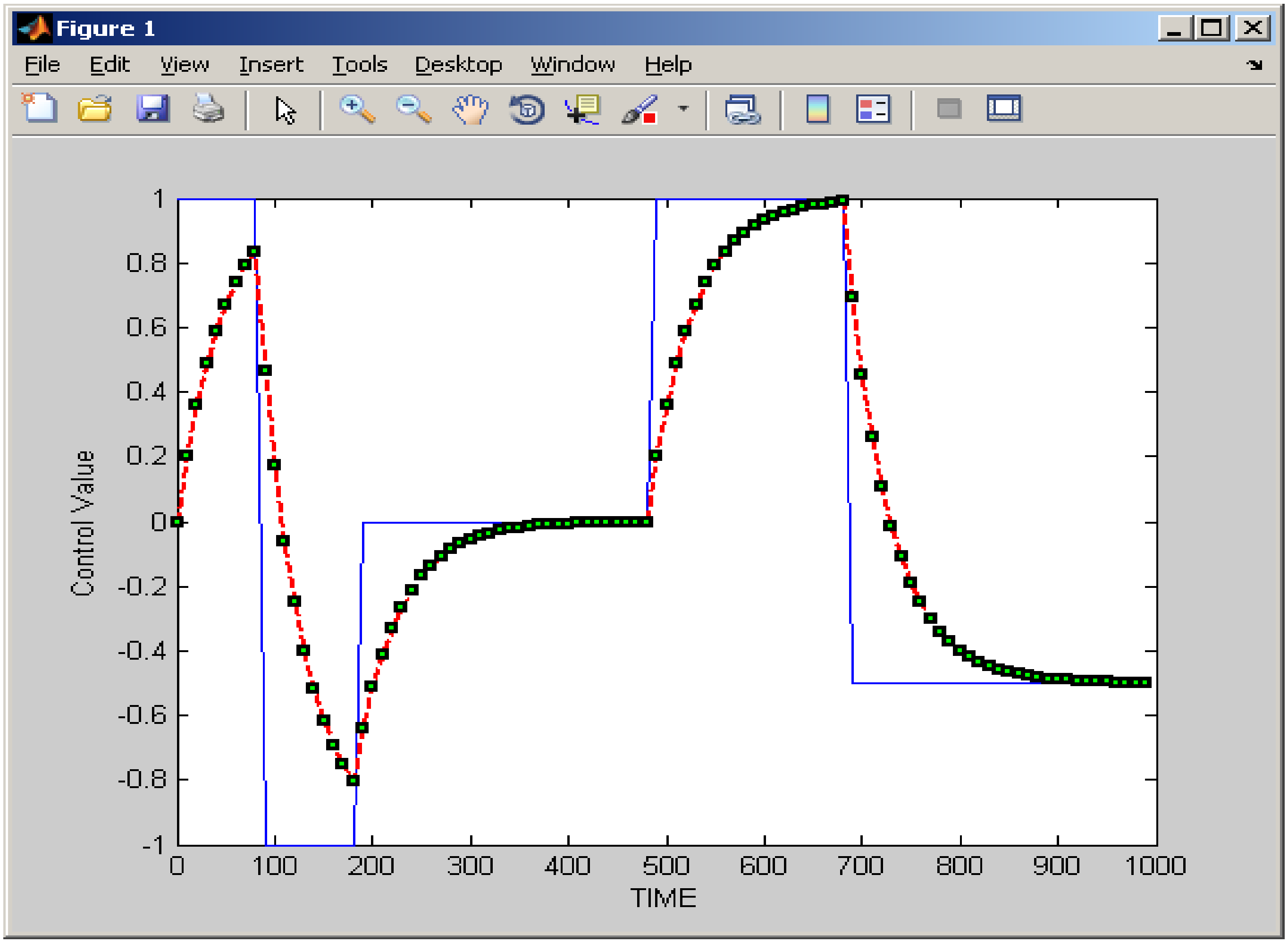


Control Low Pass Filter:

The changes in control must not be very fast instead they must be gradual. So we needed to introduce a Low pass filter in the system. The output of image processing block is dependent on the previous output and the difference between new values calculated and previous values.

New_Control_Value = Previous_Control_Value + (New_Calculated_Val - Previous_Control_Value) / GF;

Here GF is graduation factor. It decides the low pass frequency cut-off. Larger Value of GF lower the cut off frequency.



In the above diagram the blue line represents the Value of Image Processing Block and green dots represents value send by control system block to communication block after applying control low pass filter. Graduation Factor here is 5.

Problems in Control System 2.0:

The Control Low Pass Filter decreased the Reaction Time considerably. So the system was very smooth but its performance was very slow. The Low Pass filter was not dependent on time but instead it was dependent on frequency of executing the Image Processing block. An Ideal Low pass Filter would not depend on output of Image Processing Block but instead it will depend on time when the command is given.

The control value did not decayed in time so control was still little unstable if the image Processing Code crashed. The GUI was included in the capturing block so if we pressed a key to control the drone, the FPS and ardrone control thread had performance degradation.

Problem of using this control system in line following application is that the drone usually goes out of the line so it has to retrace its path to come back on the line. This feature was not included in the this control system.

Control System 3.0:

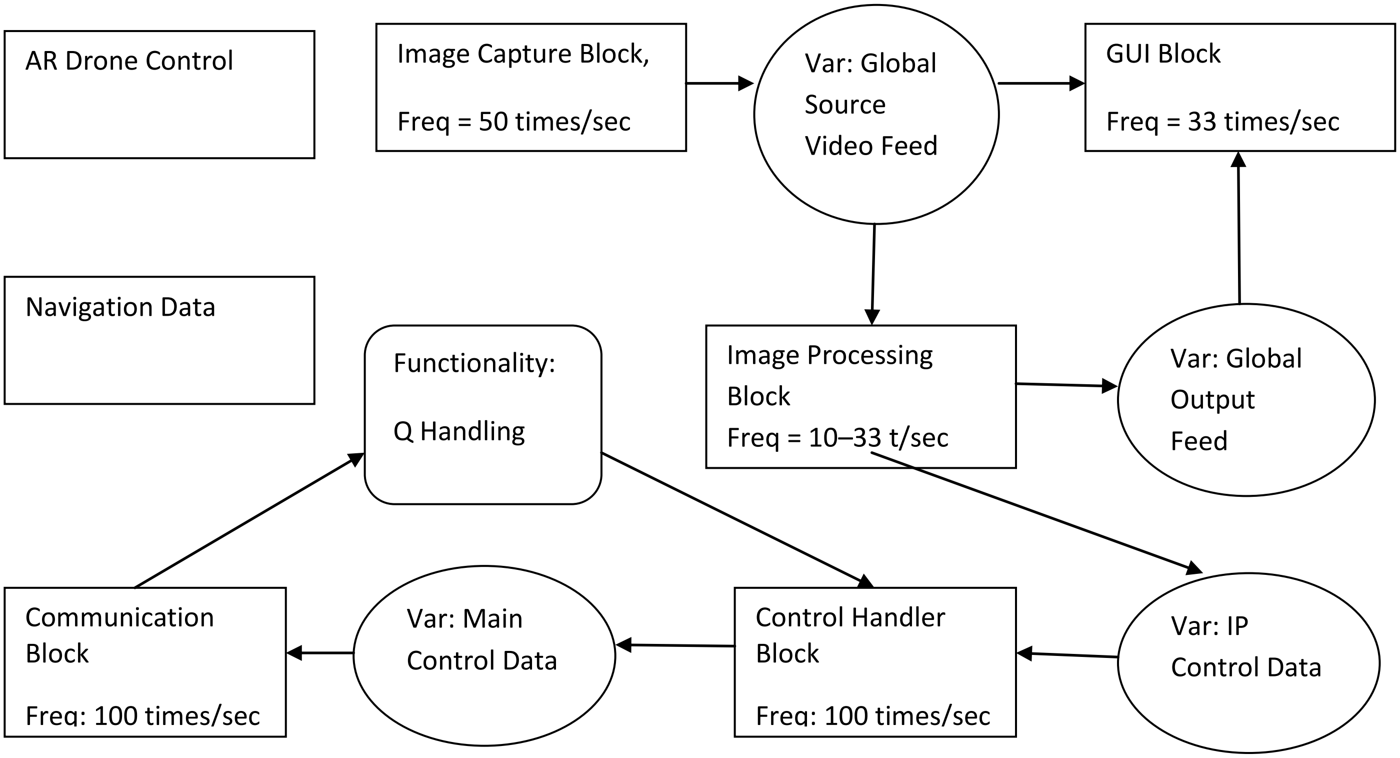
The Idea of making control system 3.0 was to make all the blocks independent of each other and they should not depend on each other's delay and other performance parameters. The blocks must only depend on time.

To achieve high flexibility in system so that all systems must depend on time we had to add a variable called 'timestamp' in all control variables so that independent blocks would know when a particular control data variable was assigned. Lots of Mutex Variables were added so that all the global variables could be accessed flexibly.

Two new blocks were added namely GUI and Control Handler. GUI was introduced so that GUI commands would not affect other performance especially FPS. Control Handler thread was used to take the control system part out of image processing thread and update the control data after every 10 milli second.

A new functionality Q was added to store all the previous commands sent to the drone and retrace the path which was followed earlier if needed.

Following is the block diagram of the system:



In above block diagram : Square Blocks indicate Threads, Circular Block Indicate Global Variables to pass data.

The term Freq is no of time in an average a particular block is executed per second.

Functionality of the Blocks :

A) Image Capture Block: Data of image is transferred to an global image variable. This variable is of OpenCV structure called IplImage.

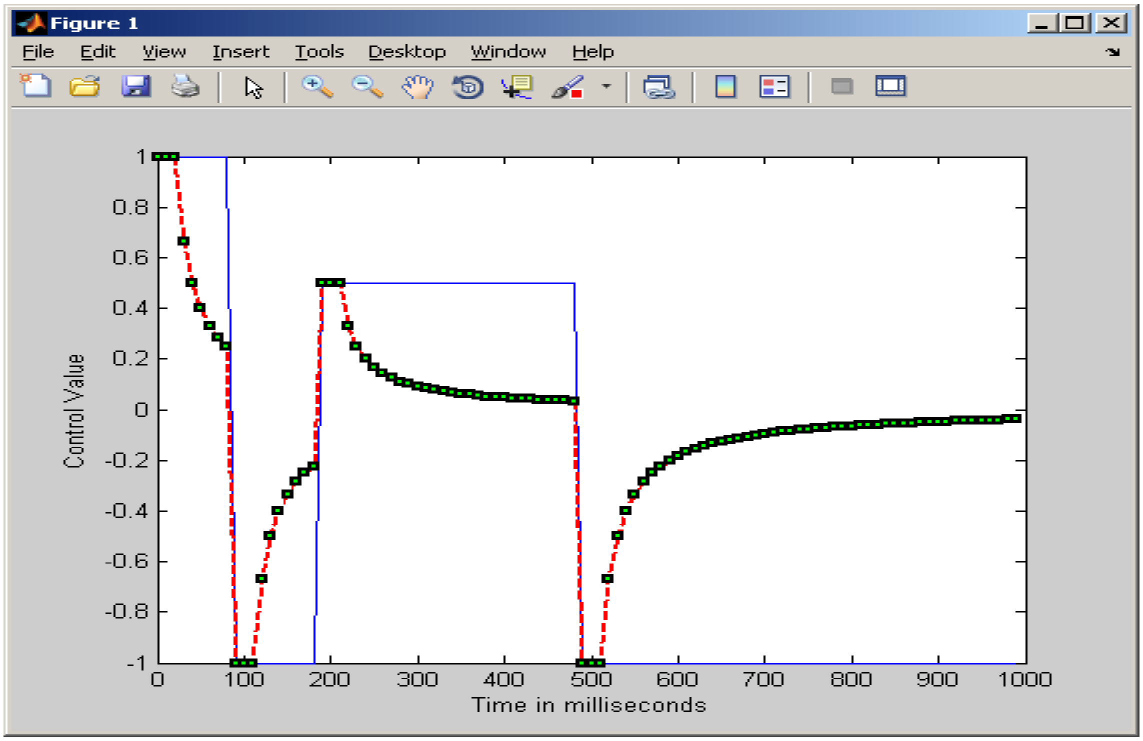
B) GUI Block: It gets the Global feed for source image and output of Image Processing Block and shows them using OpenCV functions. It also takes command from keyboard and manipulates the control functions accordingly.

C) Image Processing Block: It Processes the source image, finds out the next command that should be sent to the AR Drone. It also issues a time stamp at which it had given the following command.

D) Control Handler Block: It gets output from Image Processing Block and manipulates it according the difference in current time and the time when the control command was issued. It has high frequency of execution than Image Processing Block. This block is essential block for stability as the system is stable even if Image Processing Block Fails. The block degrades the value of the output of Image Processing Block by following formulae:

Control_Val = IP_Val / (Current_time - IP_timestamp).

Following is the Output: Here Blue line represents output of Image Processing Block and Green Dots output of Control Handler Block.



Here you can see, after 500 millisecond as the Output of Image processing Block remained constant suggesting problem in execution of Image Processing Block. Control Handler Output Decayed.

E) Communication Block: This block is used to send control data to the AR Drone. It also saves the Control Sent to the AR Drone in global Queue.

Following is the control variable structure:

```
typedef struct control_data_t                              //define the 
structure of control data;
{
    float roll; -- To store Calculated Roll Data
    float pitch; -- To store Calculated pitch Data
    float gaz; -- To store Calculated Gaz Data
    float yaw; -- To store Calculated Yaw Data
    int start; -- To identify when user wants to takeoff/land the drone
    double timestamp; --to get the time when this structure was created
}control_data_t;
```

Global Queue:

Global Queue is a functionality which helps to save the previous commands sent to the drone. Global Queue stores the output of AR Drone in Front of the Queue. If Image Processing Block suggests that we should trace our steps back, control handler starts taking the value from top of the global queue. As we don't want our computer to go out of memory control handler also removes Queue variables from bottom which are older than 5 seconds.

Implementation of global Queue:

The following are the functions which are used to implement the global queue.

```
typedef struct Q -QUEUE STRUCTURE
{
      control_data_t control;
      struct Q *next,*prev;
}Q;

void QAdd( Q **curr, control_data_t control ) -To add a contol in the globalQ
{
      //PRINT("Q_ADD\n");
      (*curr)->next = (Q *)vp_os_malloc(sizeof(Q));
      (*curr)->next->prev = (*curr);
      (*curr)=(*curr)->next;
      (*curr)->control = control;
      (*curr)->control.timestamp = cvGetTickCount();
}

Q* QInit() - To initialise the Q
{
      Q *new;
      new = (Q *)vp_os_malloc(sizeof(Q));
      new->next = NULL;
      new->prev = NULL;
      return( new );
}
```

```
void QDel( Q **curr ) – to delete a Q variable
{
      //PRINT("Q_DEL\n");
      Q *prev;
      prev = (*curr);
      (*curr)=(*curr)->next;
      (*curr)->prev = NULL;
      vp_os_free(prev);
}
```

```
control_data_t QPop( Q **curr ) – To extract latest Q variable
{
      control_data_t local_data;
      local_data = (*curr)->control;
            if ( (*curr)->prev == NULL )
            {
                  //ardrone_tool_set_ui_pad_start( 0 );
                  //ardrone_tool_shutdown_custom();
            }
      (*curr) = (*curr)->prev;
      vp_os_free( (*curr)->next );
      (*curr)->next = NULL;
      return(local_data);
}

Q *g_Qstart,*g_Qend;

void MemoryClearRoutine( )  -- Clear Q Varible if it is older than 5 seconds.
{
      double curr_time = cvGetTickCount();
      double total_time = (double)(curr_time - g_Qend->control.timestamp )/
cvGetTickFrequency();
      if(  total_time > 5000000. && g_Qend != g_Qstart )
            QDel( &g_Qend );
}
```

RESULTS:

All the results are for Control System 3.0 for application of Line Following:

Name Of Block	**Average Time Required in Block**	**Frequency of Execution**
Image Capture Block	55 ms	18.18 Hz (FPS)
GUI Block	29 ms	34.487 Hz
Control Handler Block	30 ms	33.33 Hz
Communication Block	20 ms	50 Hz
Image Processing Block (Without any Line)	50 ms	20 Hz
Image Processing Block (Line Detected)	65 ms	15.53 Hz

As Image Processing Block follows different procedure depending on whether it detects a line. So there are two different delays for each case is listed in above table. As you can see the bottle neck of system is Image Processing Block. As we increase the complexity of the block will require more time to calculate result. But the system will be stable because control handler thread helps the communication thread to decrease its output after every 30ms. The average FPS of source is 18.18 frames per second.