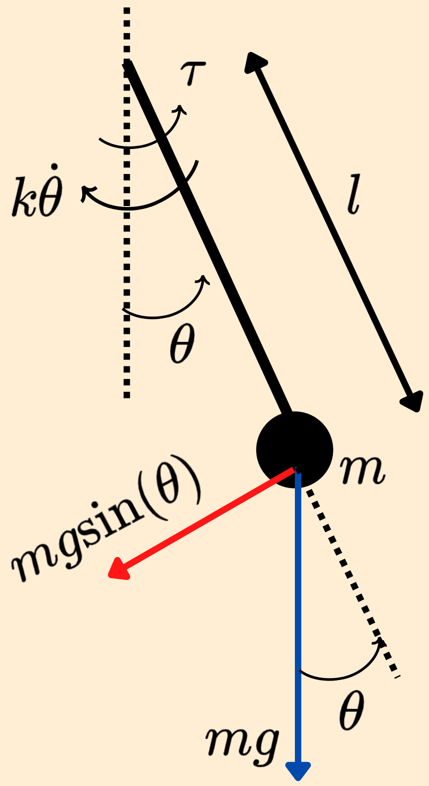
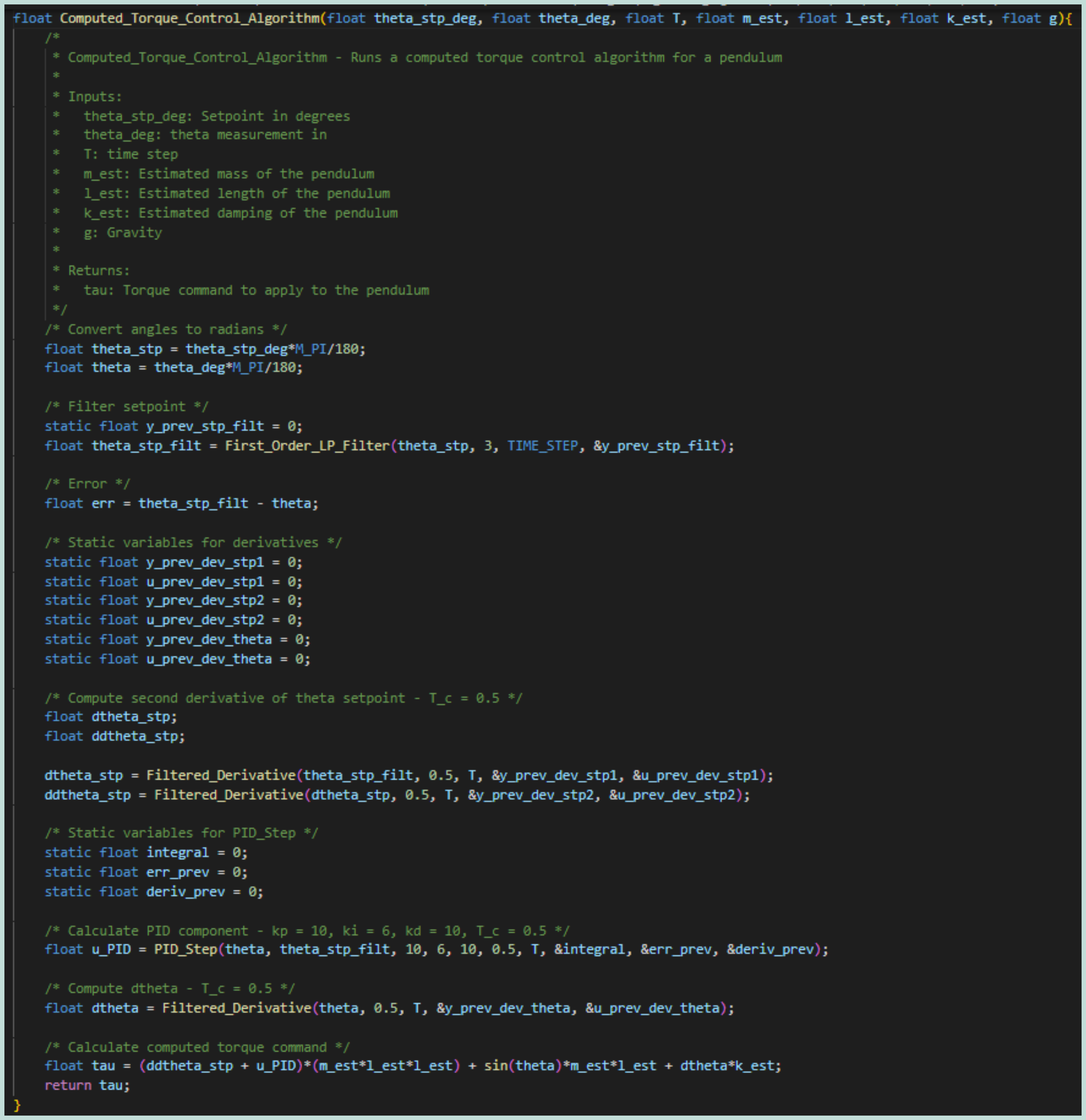
Pendulum



$$\tau = ml^2\ddot{\theta} + k\dot{\theta} + mglsin(\theta)$$

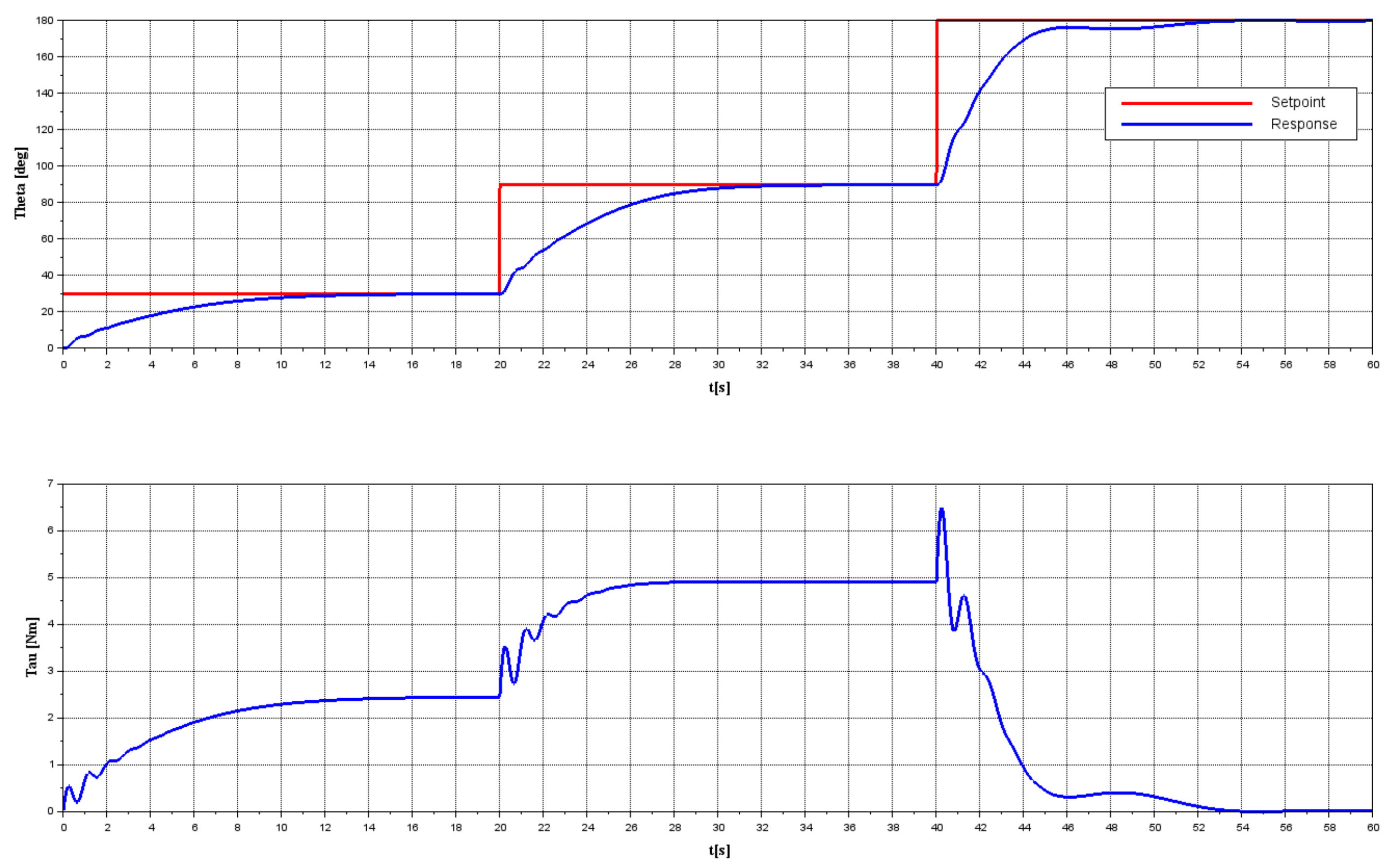
$$m = 0.5\ kg$$
$$l = 1\ m$$
$$k = 0.5\ Nms$$

Computed torque control algorithm - C function



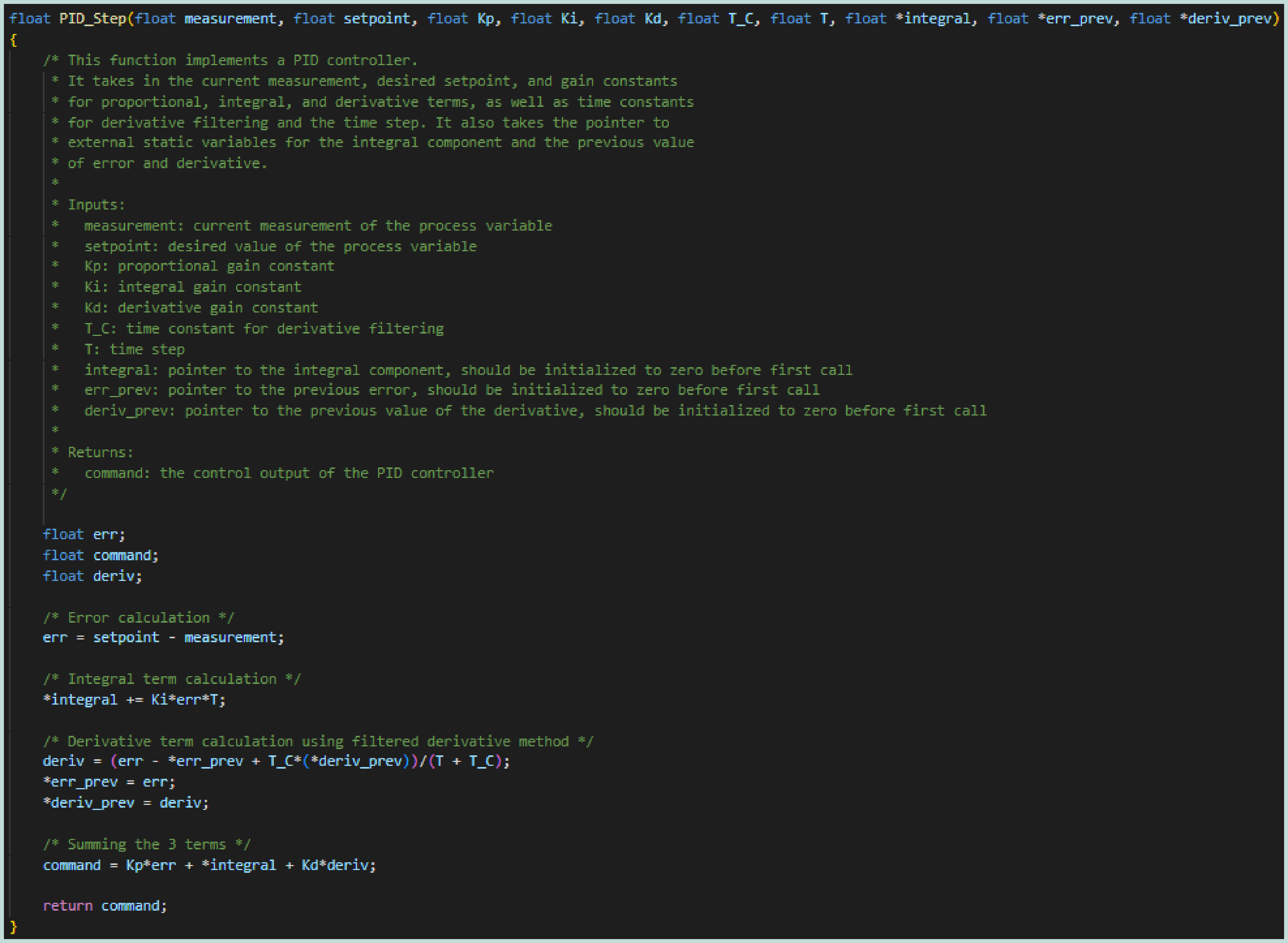
```c
float Computed_Torque_Control_Algorithm(float theta_stp_deg, float theta_deg, float T, float m_est, float l_est, float k_est, float g){
    /*
     * Computed_Torque_Control_Algorithm - Runs a computed torque control algorithm for a pendulum
     *
     * Inputs:
     *   theta_stp_deg: Setpoint in degrees
     *   theta_deg: theta measurement in
     *   T: time step
     *   m_est: Estimated mass of the pendulum
     *   l_est: Estimated length of the pendulum
     *   k_est: Estimated damping of the pendulum
     *   g: Gravity
     *
     * Returns:
     *   tau: Torque command to apply to the pendulum
     */
    /* Convert angles to radians */
    float theta_stp = theta_stp_deg*M_PI/180;
    float theta = theta_deg*M_PI/180;

    /* Filter setpoint */
    static float y_prev_stp_filt = 0;
    float theta_stp_filt = First_Order_LP_Filter(theta_stp, 3, TIME_STEP, &y_prev_stp_filt);

    /* Error */
    float err = theta_stp_filt - theta;

    /* Static variables for derivatives */
    static float y_prev_dev_stp1 = 0;
    static float u_prev_dev_stp1 = 0;
    static float y_prev_dev_stp2 = 0;
    static float u_prev_dev_stp2 = 0;
    static float y_prev_dev_theta = 0;
    static float u_prev_dev_theta = 0;

    /* Compute second derivative of theta setpoint - T_c = 0.5 */
    float dtheta_stp;
    float ddtheta_stp;

    dtheta_stp = Filtered_Derivative(theta_stp_filt, 0.5, T, &y_prev_dev_stp1, &u_prev_dev_stp1);
    ddtheta_stp = Filtered_Derivative(dtheta_stp, 0.5, T, &y_prev_dev_stp2, &u_prev_dev_stp2);

    /* Static variables for PID_Step */
    static float integral = 0;
    static float err_prev = 0;
    static float deriv_prev = 0;

    /* Calculate PID component - kp = 10, ki = 6, kd = 10, T_c = 0.5 */
    float u_PID = PID_Step(theta, theta_stp_filt, 10, 6, 10, 0.5, T, &integral, &err_prev, &deriv_prev);

    /* Compute dtheta - T_c = 0.5 */
    float dtheta = Filtered_Derivative(theta, 0.5, T, &y_prev_dev_theta, &u_prev_dev_theta);

    /* Calculate computed torque command */
    float tau = (ddtheta_stp + u_PID)*(m_est*l_est*l_est) + sin(theta)*m_est*l_est + dtheta*k_est;
    return tau;
}
```

Simulation



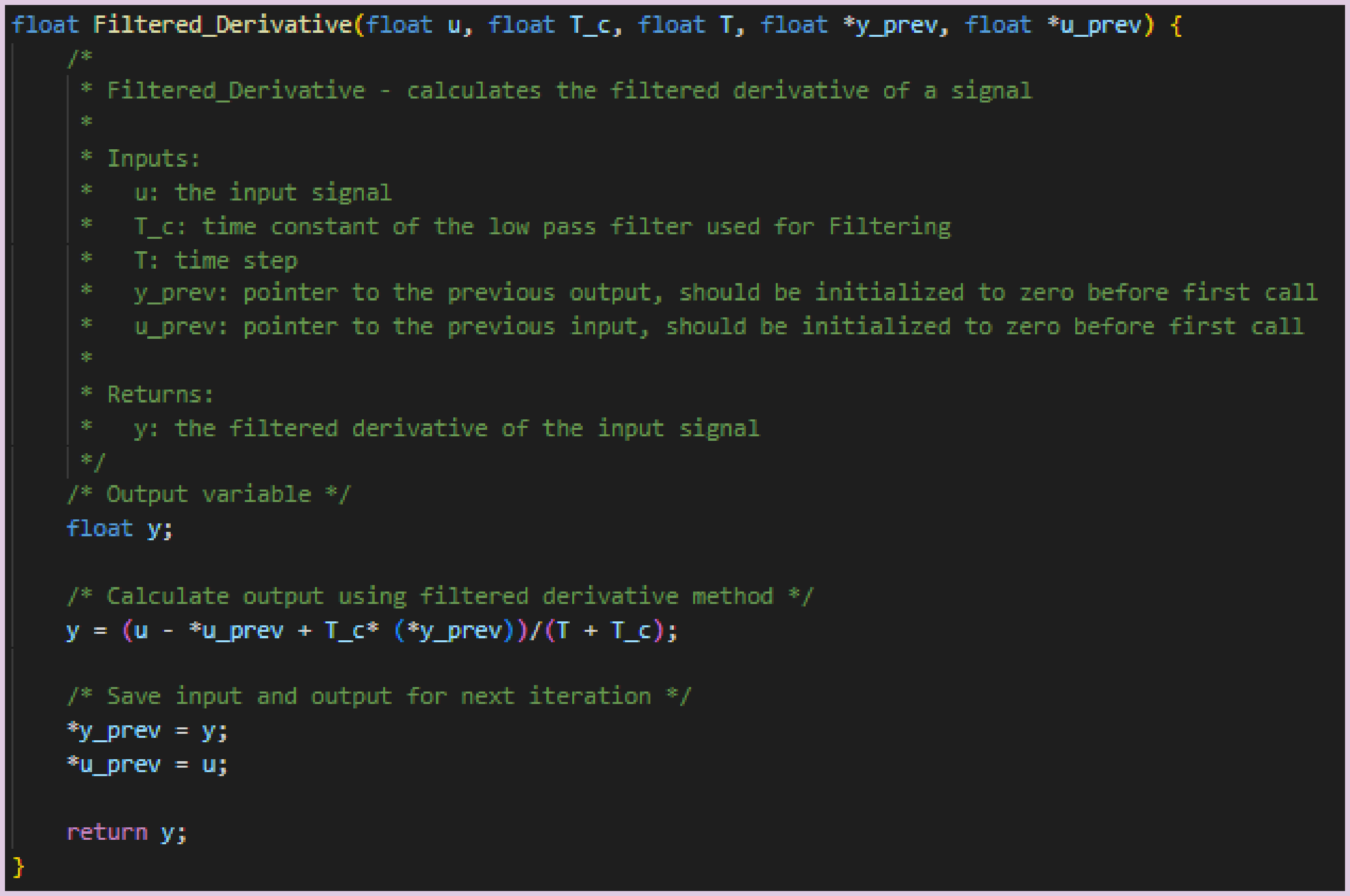
Simone Bertoni

PID step - C function



```c
float PID_Step(float measurement, float setpoint, float Kp, float Ki, float Kd, float T_C, float T, float *integral, float *err_prev, float *deriv_prev)
{
    /* This function implements a PID controller.
     * It takes in the current measurement, desired setpoint, and gain constants
     * for proportional, integral, and derivative terms, as well as time constants
     * for derivative filtering and the time step. It also takes the pointer to
     * external static variables for the integral component and the previous value
     * of error and derivative.
     *
     * Inputs:
     *   measurement: current measurement of the process variable
     *   setpoint: desired value of the process variable
     *   Kp: proportional gain constant
     *   Ki: integral gain constant
     *   Kd: derivative gain constant
     *   T_C: time constant for derivative filtering
     *   T: time step
     *   integral: pointer to the integral component, should be initialized to zero before first call
     *   err_prev: pointer to the previous error, should be initialized to zero before first call
     *   deriv_prev: pointer to the previous value of the derivative, should be initialized to zero before first call
     *
     * Returns:
     *   command: the control output of the PID controller
     */

    float err;
    float command;
    float deriv;

    /* Error calculation */
    err = setpoint - measurement;

    /* Integral term calculation */
    *integral += Ki*err*T;

    /* Derivative term calculation using filtered derivative method */
    deriv = (err - *err_prev + T_C*(*deriv_prev))/(T + T_C);
    *err_prev = err;
    *deriv_prev = deriv;

    /* Summing the 3 terms */
    command = Kp*err + *integral + Kd*deriv;

    return command;
}
```

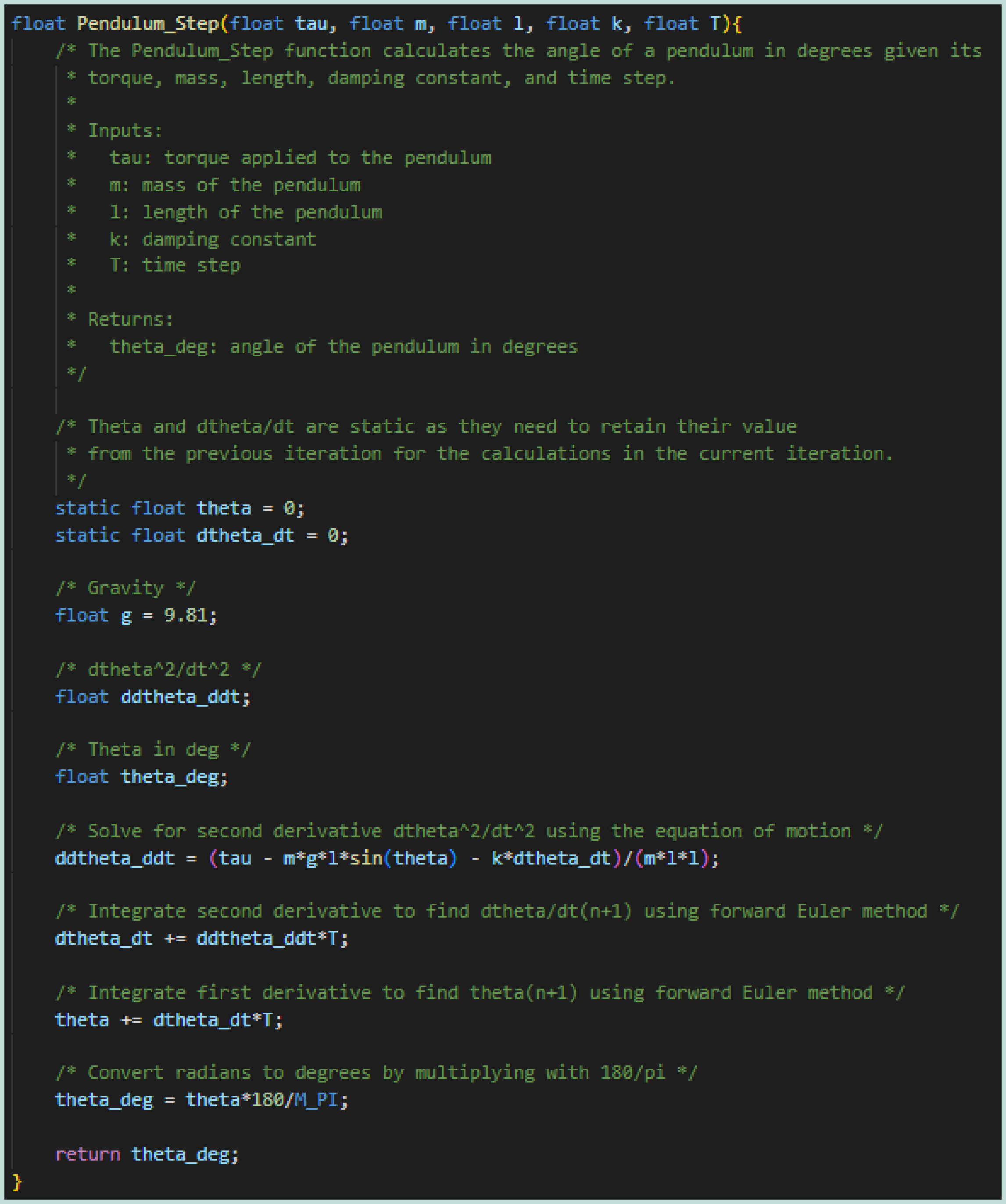
Filtered derivative - C function



```c
float Filtered_Derivative(float u, float T_c, float T, float *y_prev, float *u_prev) {
    /*
     * Filtered_Derivative - calculates the filtered derivative of a signal
     *
     * Inputs:
     *   u: the input signal
     *   T_c: time constant of the low pass filter used for Filtering
     *   T: time step
     *   y_prev: pointer to the previous output, should be initialized to zero before first call
     *   u_prev: pointer to the previous input, should be initialized to zero before first call
     *
     * Returns:
     *   y: the filtered derivative of the input signal
     */
    /* Output variable */
    float y;

    /* Calculate output using filtered derivative method */
    y = (u - *u_prev + T_c* (*y_prev))/(T + T_c);

    /* Save input and output for next iteration */
    *y_prev = y;
    *u_prev = u;

    return y;
}
```

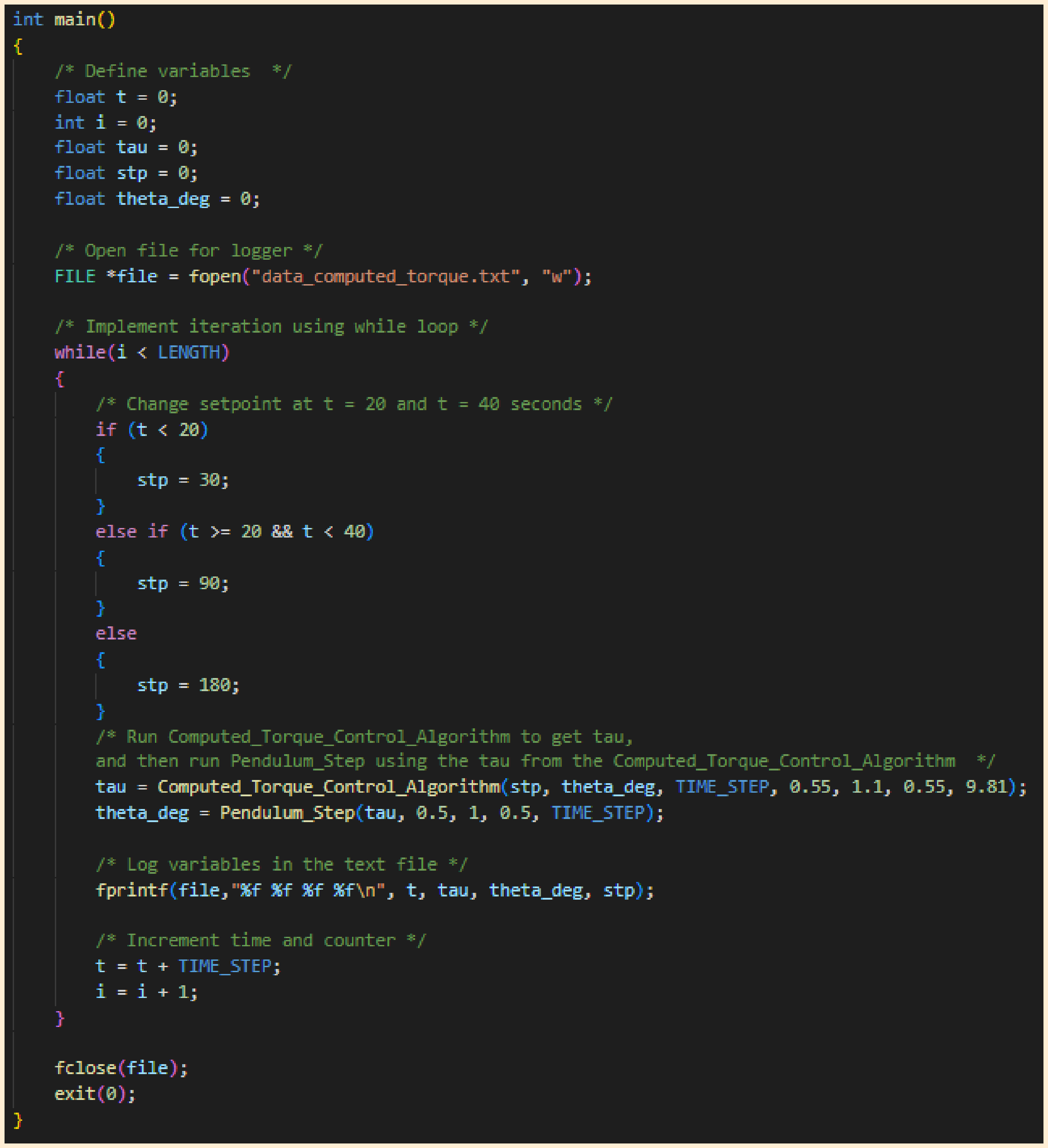
Simone Bertoni

Pendulum step - C function



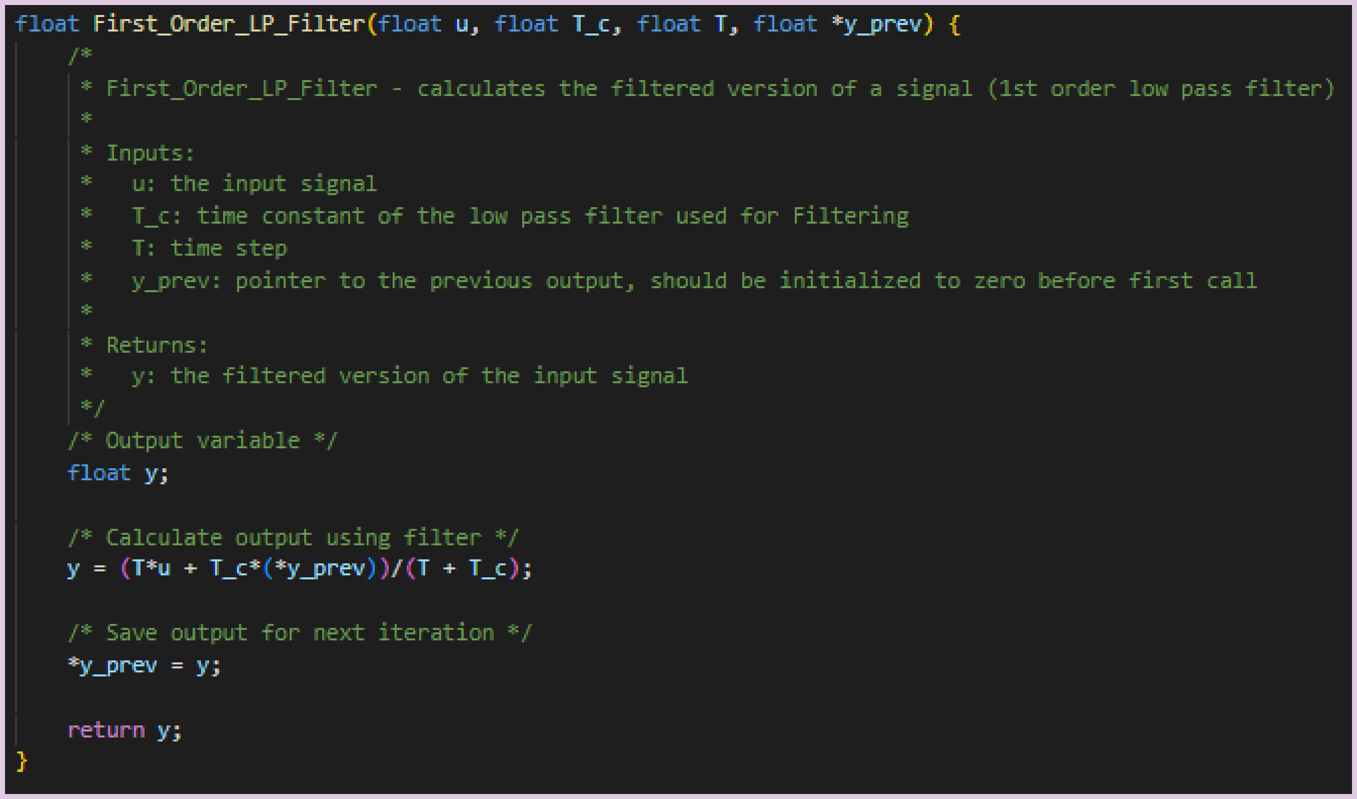
```c
float Pendulum_Step(float tau, float m, float l, float k, float T){
    /* The Pendulum_Step function calculates the angle of a pendulum in degrees given its
     * torque, mass, length, damping constant, and time step.
     *
     * Inputs:
     *   tau: torque applied to the pendulum
     *   m: mass of the pendulum
     *   l: length of the pendulum
     *   k: damping constant
     *   T: time step
     *
     * Returns:
     *   theta_deg: angle of the pendulum in degrees
     */

    /* Theta and dtheta/dt are static as they need to retain their value
     * from the previous iteration for the calculations in the current iteration.
     */
    static float theta = 0;
    static float dtheta_dt = 0;

    /* Gravity */
    float g = 9.81;

    /* dtheta^2/dt^2 */
    float ddtheta_ddt;

    /* Theta in deg */
    float theta_deg;

    /* Solve for second derivative dtheta^2/dt^2 using the equation of motion */
    ddtheta_ddt = (tau - m*g*l*sin(theta) - k*dtheta_dt)/(m*l*l);

    /* Integrate second derivative to find dtheta/dt(n+1) using forward Euler method */
    dtheta_dt += ddtheta_ddt*T;

    /* Integrate first derivative to find theta(n+1) using forward Euler method */
    theta += dtheta_dt*T;

    /* Convert radians to degrees by multiplying with 180/pi */
    theta_deg = theta*180/M_PI;

    return theta_deg;
}
```

Main - C function



```c
int main()
{
    /* Define variables  */
    float t = 0;
    int i = 0;
    float tau = 0;
    float stp = 0;
    float theta_deg = 0;

    /* Open file for logger */
    FILE *file = fopen("data_computed_torque.txt", "w");

    /* Implement iteration using while loop */
    while(i < LENGTH)
    {
        /* Change setpoint at t = 20 and t = 40 seconds */
        if (t < 20)
        {
            stp = 30;
        }
        else if (t >= 20 && t < 40)
        {
            stp = 90;
        }
        else
        {
            stp = 180;
        }
        /* Run Computed_Torque_Control_Algorithm to get tau,
        and then run Pendulum_Step using the tau from the Computed_Torque_Control_Algorithm  */
        tau = Computed_Torque_Control_Algorithm(stp, theta_deg, TIME_STEP, 0.55, 1.1, 0.55, 9.81);
        theta_deg = Pendulum_Step(tau, 0.5, 1, 0.5, TIME_STEP);

        /* Log variables in the text file */
        fprintf(file,"%f %f %f %f\n", t, tau, theta_deg, stp);

        /* Increment time and counter */
        t = t + TIME_STEP;
        i = i + 1;
    }

    fclose(file);
    exit(0);
}
```

First order low pass filter - C function



```c
float First_Order_LP_Filter(float u, float T_c, float T, float *y_prev) {
    /*
     * First_Order_LP_Filter - calculates the filtered version of a signal (1st order low pass filter)
     *
     * Inputs:
     *   u: the input signal
     *   T_c: time constant of the low pass filter used for Filtering
     *   T: time step
     *   y_prev: pointer to the previous output, should be initialized to zero before first call
     *
     * Returns:
     *   y: the filtered version of the input signal
     */
    /* Output variable */
    float y;

    /* Calculate output using filter */
    y = (T*u + T_c*(*y_prev))/(T + T_c);

    /* Save output for next iteration */
    *y_prev = y;

    return y;
}
```

Simone Bertoni