ECE MBD: TU4 p2

Self-balancing bot: Rapid Control Prototyping (RCP)

Based on the previous laboratory session (TU4 p1) and the system knowledge gained, the model based developed shall be continued and the prepared and tested control algorithm shall be used for the control of a real system.

The hardware consists of an Arduino Mega 2650 and a motor/sensor shield MPU6050. To access the data from the sensor shield and to operate the actuators it is required to configure the corresponding registers via I2C. Within the model environment this will be done by the one-time initialization subsystem block. After the configuration of the sensor shield the sensors and actuators shall be taken into operation. A first step is to check the response of the accelerometer, followed by the operation of the motors.

By using the external mode of Simulink it will be possible to interact with the hardware. That allows you to read the sensor information and write operate actuators online.

Figure 1: Self-balancing bot

Laboratorywork

Model Preparation

1. Prepare a new model in your MATLAB working directory and configure it as follows (or use the template provided "selfbalancingbot_2019a_template.slx"):
    - Select a proper target by setting up your model in the Configuration Parameters Panel/Hardware Implementation to the Arduino Mega 2560. Remember the sequence lock unit
      The detection of the COM should work automatically; but in case that it does not, then change the Host-board connection mode to Manual and then specify the COM port of your serial communication.
    - Set the fixed-step solver of your model to 10ms (remember NF1 of previous lab).

2. The Simulink Model shall have the following structure:

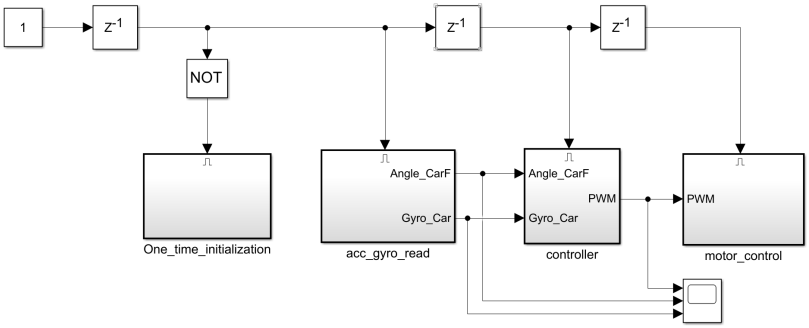

Figure 2: Model framework

The enabled subsystems represent a consecutive execution of the implemented functions. The first subsystem is the initialization of the sensors and outputs and it is executed only once when the model is launched.

System Initialization (MPU6050)

3. In order to operate the MPU6050 it has to be configured first, therefore the registers of the board have to be set using the I2C and Digital output blocks from the Simulink Arduino library.
    - Implement the initialization sequence as shown below. An example of how to configure the I2C block is shown in Figure 4. The corresponding slave device address is 0x68

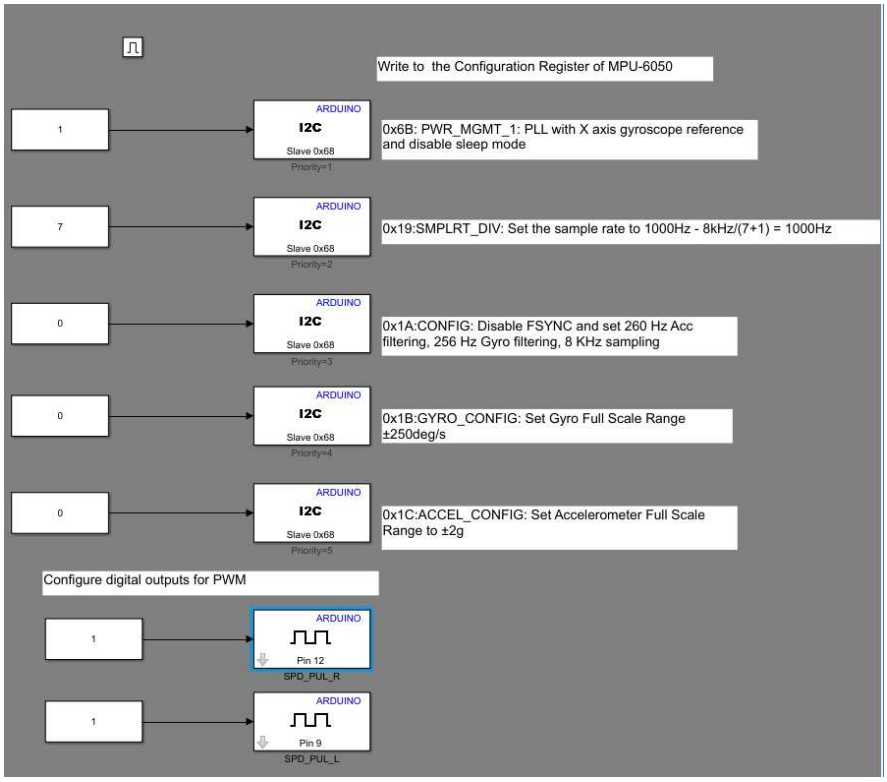

Figure 3: One time initialization

A comprehensive list of the MPU 6050 and the registers which are configured can be found at: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

Here is an example of the first block:

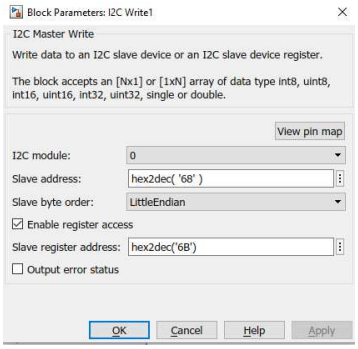

Figure 4: I2C Block Configuration

Note: For your information, the model above implements the following C code:

```
  i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
  i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz
Gyro filtering, 8 KHz sampling
  i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
  i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g

  while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers
at once
  while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope reference
and disable sleep mode
```

Read Gyro Sensor Data

4. The second subsystem (model framework) shall read out the sensor values from the target at a 10ms rate and compute the angular speeds and accelerations; therefore:
   - Use the I2C blocks to read out the sensor data (as shown below)
   - The information from the I2C has to be converted into usable data. Implement the two calculation (once for the angle and accelerator) blocks as MATLAB functions. The Function(s) for gathering the angle and acceleration can be found as an annex.

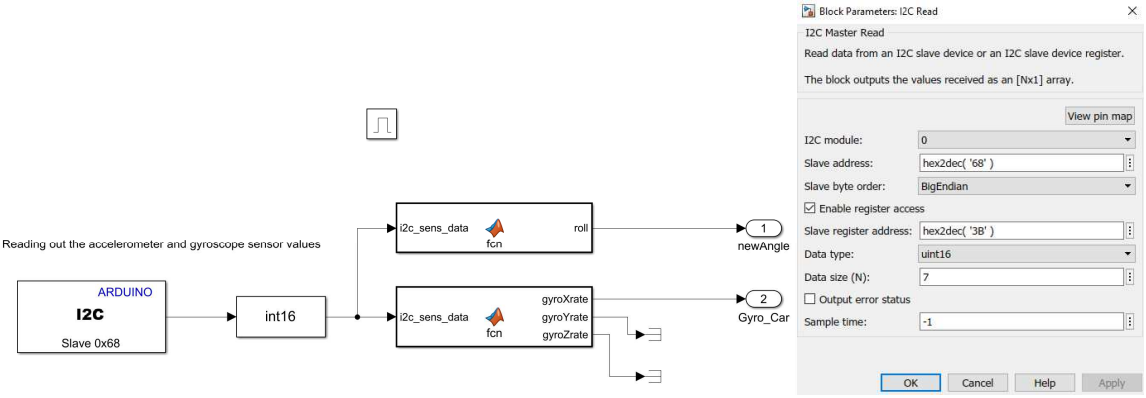

Figure 5: I2C Sensor Configuration

5. Test the implemented functions by operating/running the model in external mode. Analyze/observe the signals provided by the sensor using a scope.
6. Attach the Kalman Filter according to FR1 (see Annex) to the gathered signals (roll to newAngle, gyroXrate to newRate, dt shall be the sample time) of Figure 5
    - Make a comparison between the raw values and the filtered angle, report your findings

Actuators (motors)

7. The fourth subsystem will write the digital signals which are used to control the two motors. Use the digital write blocks from the Simulink Arduino library to implement the motor control.
    - The speed of the motor is configured via PWM blocks from the library (max. 255)
    - The rotational direction of the motor is configured via digital pins see Figure 6.
    - Test your setup and the operation of the actuators in External Mode with the Balanbot
      For the final setup, consider that the drive direction shall be based on the actual control command
    - Recommendations for testing:
        - In order to test the algorithm, integrate a slider gain for the PWM as it allows you to change the speed values of the motors in external mode.
        - Use a manual switch in the Simulink model to switch the motor drive direction during external mode simulation.

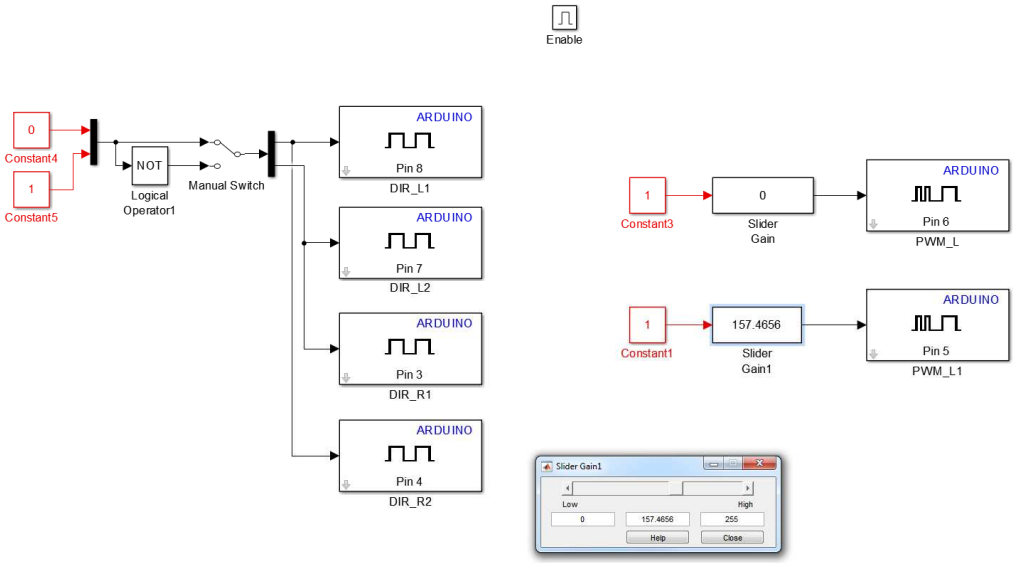

Figure 6: Motor control

Controller

8. Integrate the third main block (controller) developed in the previous Lab.
   Remember NF3:
   - The proportional, integral and derivative gains (Kp, Ki and Kd; respectively) shall be adjustable during run
   - Kp shall be multiplied by angle (output of Kalman filter)
   - Ki shall be multiplied by the integral of angle (output of Kalman filter)
   - **Kd shall be multiplied directly by gyroXrate** (newRate input of the Kalman filter)
     - Note: In the hardware implementation this is a reliable signal coming from the gyroscope
   - The output of the controller (sum of the three previous parts) shall be saturated to -255 and 255
9. Integrate the FR2 which shall operate the motors only if the angle is in the range between -40° and 40° otherwise the motor operation shall stop.
10. Tune the controller for achieving a stable balanbot
    - Report your findings

From the exercises, it is expected that you report

- Your development process: design and testing
- Add a short video of the working Balanbot

Organization:

The Balanbots are coordinated by Mr. Läßer and **have to** be returned to him after finishing the tasks!!!

Important:

Consider to use always the same Balanbot for your test, another Balanbot may react different as the hardware may have a different motor reaction, gear plays etc.

Annex:

1. Reading of Gyro Data

```
function roll = fcn(i2c_sens_data)
accX = double(i2c_sens_data(1));
accY = double(i2c_sens_data(2));
accZ = double(i2c_sens_data(3));
roll  = atan(accY / sqrt(accX * accX + accZ * accZ)) * 180/pi;

function [gyroXrate,gyroYrate,gyroZrate] = fcn(i2c_sens_data)
gyroX = double(i2c_sens_data(5));
gyroY = double(i2c_sens_data(6));
gyroZ = double(i2c_sens_data(7));
gyroXrate = gyroX/131;
gyroYrate = gyroY/131;
gyroZrate = gyroZ/131;
```

2. Kalman Filter

```
function [angle,bias] = KalmanFil_Balanbot(newAngle,newRate,dt)
         % This script was adapted for its implementation in Simulink by R. Estrada (FH
         JOANNEUM 2017)
               % Based on KasBot V2  -  Kalman filter module - http:%www.x-
               firm.com/?page_id=145
        % Modified by Kristian Lauszus
         % More information: http:%blog.tkjelectronics.dk/2012/09/a-practical-approach-to-
        kalman-filter-and-how-to-implement-it

        % Discrete Kalman filter time update equations - Time Update ("Predict")
        % Update xhat - Project the state ahead

        coder.extrinsic('exist');

        % Declaration of persistent in memory variables (similar to declare global
        % variables)

        persistent init0 P K x;

        % Initialization of typical sensor and noise variances
        Q_angle = 0.001;
        Q_bias = 0.003;
        R_measure = 0.03;

        % Declaration/initialization of main static matrices and vectors
        F = [1      -dt; 0      1];     % Transition matrix
        Q = [Q_angle  0; 0 Q_bias];     % Process noise covariance matrix
        B = [dt ; 0];                   % Control-input model
        H = [1    0];                   % Observation model

        % SETUP: Executed only the first time
        if isempty(init0)
            init0=0;
             % Declaration/initialization of main dynamic matrix and vectors
            P     = [0 0;0 0];  % Error covariance matrix
            K     = [0;0];      % Kalman gain vector
            x     = [0;0];      % State vector
        end

        % LOOP: Implementation of Kalman filter algorithm

        % Step 1 */
        % Calculate prior state vector
        x = F * x + B * newRate;

        % Step 2 */
        % Update estimation error covariance - Project the error covariance ahead
         P = F * P * F' + Q * dt;

        % Step 3 */
        % Update estimate with measurement zk (newAngle)
        y = newAngle - H * x;
```

```
% Step 4 */
% Discrete Kalman filter measurement update equations - Measurement Update ("Correct")
S = H * P * H' + R_measure;

% Step 5 */
% Compute the Kalman gain
K = P * H' / S;

% Step 6 */
% Update state vector
x = x + K * y;

% Step 7 */
% Calculate estimation error covariance - Update the error covariance
P = (eye(2) - K * H) * P;

% RETURN: Update the function outputs
angle = x(1);
bias  = x(2);
```