Technische Universität Berlin
Institute for Process an Chemical Engineering
Chair of Control Engineering

IESS – Laboratory 4

Observer design for the MinSeg™ self-balancing robot

In this laboratory, we will design a Kalman filter for a self-balancing robot using gyroscope and accelerometer data from the MinSeg™'s on-board MPU6050 (Inertial Measurement Unit (IMU)). A Kalman filter is a powerful tool for estimating the state of a system based on noisy measurements. In the case of a self-balancing robot, the gyroscope and accelerometer data provide information on the robot's orientation and motion; however, these measurements are subject to noise and uncertainty, which can make it challenging to accurately estimate the robot's state.

In this laboratory document, we will guide you through the process of designing a Kalman filter to estimate the state of a self-balancing robot based on gyroscope and accelerometer data. We will begin by collecting gyroscope and accelerometer data. We will then walk through the process of converting our continuous-time model to a discrete-time model. Then, we will introduce the basics of Kalman filtering and how to use the IMU data to estimate the state of the robot. Finally, we will provide some tips and best practices for tuning the Kalman filter to optimize its performance. By the end of this laboratory, you will have gained a deeper understanding of the Kalman filter and its application to robotics, as well as the skills and knowledge necessary to design and implement a Kalman filter for a self-balancing robot.

Task 1 – Collecting IMU data

1. Connect your MinSeg™ robot to Ardunio and collect data from the IMU using the code in Listing 1 while moving the robot around and then holding it at a 5-degree angle for some time.
   Note: The measurements will be printed to your serial monitor in Arduino and you will need to copy the values into a text file or excel document.

2. Load your data into your MATLAB workspace.

Listing 1: Arduino code for collecting gyroscope and accelerometer measurement data from an MPU6050. This code initializes the MPU6050 sensor and sets the full-scale range for the gyroscope and accelerometer measurements. The gyroscope measurements are converted from raw values to degrees per second using the 65.5 factor, while the accelerometer measurements are converted from raw values to meters per second squared using the 16384.0 factor.

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  mpu.initialize(); //Initializes MPU for measurements +/-250dps and
      +/-2g
}

void loop() {
  int16_t ax, ay, az, gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  Serial.print("Accelerometer␣(m/s^2):␣");
  Serial.print(ax/16384.0); Serial.print(",␣"); //Scale with 2^-14
     to convert readings into +/- 2g
  Serial.print(ay/16384.0); Serial.print(",␣");
  Serial.println(az/16384.0);
  Serial.print("Gyroscope␣(deg/s):␣");
  Serial.print(gx/131.0); Serial.print(",␣"); //Scale with 250/2^15
     to convert readings into +/- 250dps
  Serial.print(gy/131.0); Serial.print(",␣");
  Serial.println(gz/131.0);
  delay(1000);
}
```

State-space discreteization

For a continuous-time state-space system

$$\dot{x} = Ax + Bu \tag{1}$$
$$y = Cx + Du\,, \tag{2}$$

we obtain the discretized system

$$x_{k+1} = A_d x_k + B_d u_k \tag{3}$$
$$y_k = C_d x_k + D_d u_k\,, \tag{4}$$

where

$$A_d = e^{AT} = \mathcal{L}^{-1}\Big\{(sI - A)^{-1}\Big\}_{t-T} \tag{5}$$
$$B_d = \left(\int_0^T e^{A\tau}\mathrm{d}\tau\right) B = A^{-1}(A_d - I)B\,, \quad \text{if } A \text{ is non-singular} \tag{6}$$
$$C_d = C \tag{7}$$
$$D_d = D\,, \tag{8}$$

and e^{AT} is a matrix exponential.

Kalman Filter

The Kalman filter is a mathematical algorithm that is widely used in control systems and signal-processing applications to estimate the state of a system from noisy or incomplete measurements. The basic idea behind the Kalman filter is to use a recursive least squares approach to estimate the system state and update the estimates based on incoming measurements.

Suppose we have a system that can be described by the following discrete-time state-space equations:

$$x_{k+1} = A_d x_k + B_d u_k + w_k \tag{9}$$

$$y_k = C_d x_k + D_d u_k + v_k \,, \tag{10}$$

where x_k is the state vector, u_k is the input vector, y_k is the measurement vector, and $w_k \sim \mathcal{N}(0,Q_d)$ is the process noise and $v_k \sim \mathcal{N}(0,R_d)$ is the measurement noise. The matrices A_d, B_d, C_d, and D_d are the system matrices that describe the behavior of the system.

The goal of the Kalman filter is to estimate the state vector $\hat{x}_k \sim \mathcal{N}(\mu_{k|k},P_{k|k})$ given the measurement vector y_k. To do this, the Kalman filter uses a recursive algorithm that updates the estimate of the state vector at each time step. The algorithm consists of two steps: the prediction step and the update step.

The Kalman filter can be described mathematically using the following equations:

Prediction:

State prediction:

$$\mu_{k|k-1} = A_d \mu_{k-1|k-1} + B_d u_{k-1} \,, \tag{11}$$

where $\mu_{k|k-1}$ is the predicted state of the system at time k, given the previous estimated state $\mu_{k-1|k-1}$.

Error covariance prediction:

$$P_{k|k-1} = A_d P_{k-1|k-1} A_d^T + Q_d \,, \tag{12}$$

where $P_{k|k-1}$ is the predicted error covariance matrix at time k, given the error covariance matrix at time $k-1$,

Update:

Kalman gain:

$$K_k = P_{k|k-1} C_d^T (C_d P_{k|k-1} C_d^T + R_d)^{-1} \,, \tag{13}$$

where K_k is the Kalman gain matrix.

State update:

$$\mu_{k|k} = \mu_{k|k-1} + K_k (y_k - C_d \mu_{k|k-1} - D_d u_k) \,, \tag{14}$$

where $\mu_{k|k}$ is the predicted state updated by the measurement y_k.

Error covariance update:

$$P_{k|k} = (I - K_k C_d) P_{k|k-1} \,. \tag{15}$$

Task 2 – Kalman filter from IMU data

We use the gyroscope measurements to obtain the chassis angular velocity, $\omega = \dot{\theta}$, and measurement bias, b. Additionally, we use the accelerometer measurements to estimate the angle of the chassis, θ. From this, we obtain the continuous time process model

$$\begin{bmatrix} \dot{\omega}(t) \\ \dot{\theta}(t) \\ \dot{b}(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \omega(t) \\ \theta(t) \\ b(t) \end{bmatrix} + \begin{bmatrix} \dot{w}_{\dot{\theta}}(t) \\ 0 \\ \dot{w}_b(t) \end{bmatrix} ,$$

with covariance

$$Q = \begin{bmatrix} q_\omega & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & q_b \end{bmatrix} .$$

Then the discrete-time process model is given by

$$A_d = e^{AT} = \begin{bmatrix} 1 & 0 & 0 \\ T & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} ,$$

to give the discrete-time model

$$\begin{bmatrix} \omega_k \\ \theta_k \\ b_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ T & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{k-1} \\ \theta_{k-1} \\ b_{k-1} \end{bmatrix} + \begin{bmatrix} w_{\omega_k} \\ 0 \\ w_{b_k} \end{bmatrix} ,$$

with covariance

$$Q_d = \int_0^T e^{A\tau} Q e^{A^\top \tau} \mathrm{d}\tau = \begin{bmatrix} Tq_\omega & \frac{1}{2}T^2 q_\omega & 0 \\ \frac{1}{2}T^2 q_\omega & \frac{1}{3}T^3 q_\omega & 0 \\ 0 & 0 & Tq_b \end{bmatrix} .$$

The measurement model uses accelerometer measurements in the radial and tangential directions; as shown in Figure 1; to estimate θ, where translational accelerations are neglected and zero-mean white noise is added as measurement noise. This results in the non-linear approximation

$$y_\theta(t) = \arctan 2 \left(\frac{-a_r(t) + \dot{v}_r(t)}{a_t(t) + \dot{v}_t(t)} \right) \approx \theta(t) + \dot{v}_\theta(t) .$$

Gyroscopic measurements include the angular velocity and bias states with the zero-mean white noise of

$$y_\omega(t) = \omega(t) + b(t) + \dot{v}_\omega(t) .$$

This gives the full continuous-time measurement model

$$\begin{bmatrix} y_\theta(t) \\ y_\omega(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega(t) \\ \theta(t) \\ b(t) \end{bmatrix} + \begin{bmatrix} \dot{v}_\theta(t) \\ \dot{v}_\omega(t) \end{bmatrix} ,$$

with covariance

$$R = \begin{bmatrix} r_\theta & 0 \\ 0 & r_\omega \end{bmatrix} .$$

Where the discrete-time measurement model is

$$\begin{bmatrix} y_{\theta_k} \\ y_{\omega_k} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_k \\ \theta_k \\ b_k \end{bmatrix} + \begin{bmatrix} v_{\theta_k} \\ v_{\omega_k} \end{bmatrix} ,$$

with covariance

$$R_d = \frac{1}{T} \begin{bmatrix} r_\theta & 0 \\ 0 & r_\omega \end{bmatrix} .$$

The Kalman filter tuning parameters are the $p_{\cdot}$, $q_{\cdot}$, and $r_{\cdot}$ values in the P, Q_d, and R_d matrices. We give a guide for tuning these parameters in Table 1.

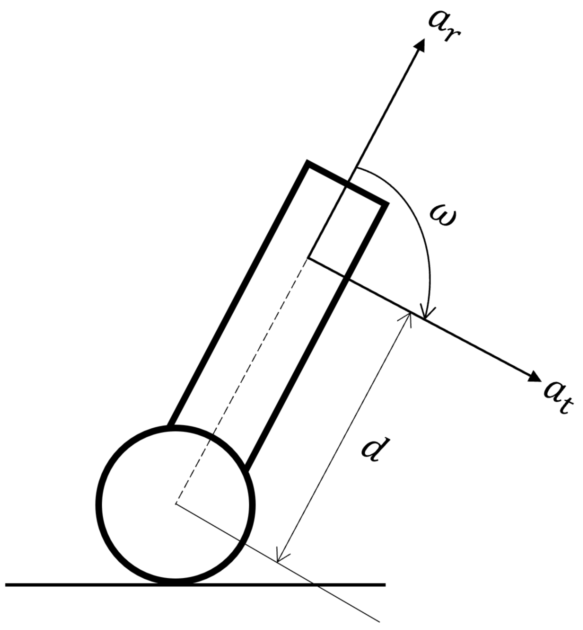

Figure 1: Schematic of the gyroscope mount of the two-axis gyroscope.

1. Make a MATLAB script to initialize the tuning parameters of the Kalman filter as global variables.

2. Use the algorithm in Figure 2 to implement the Kalman filter to estimate the angular velocity and tilt angle of the chassis from the IMU data.

3. Implement the Kalman filter algorithm on the Arduino to obtain real-time estimates of the chassis angular velocity $\dot{\theta}$ and tilt angle θ.

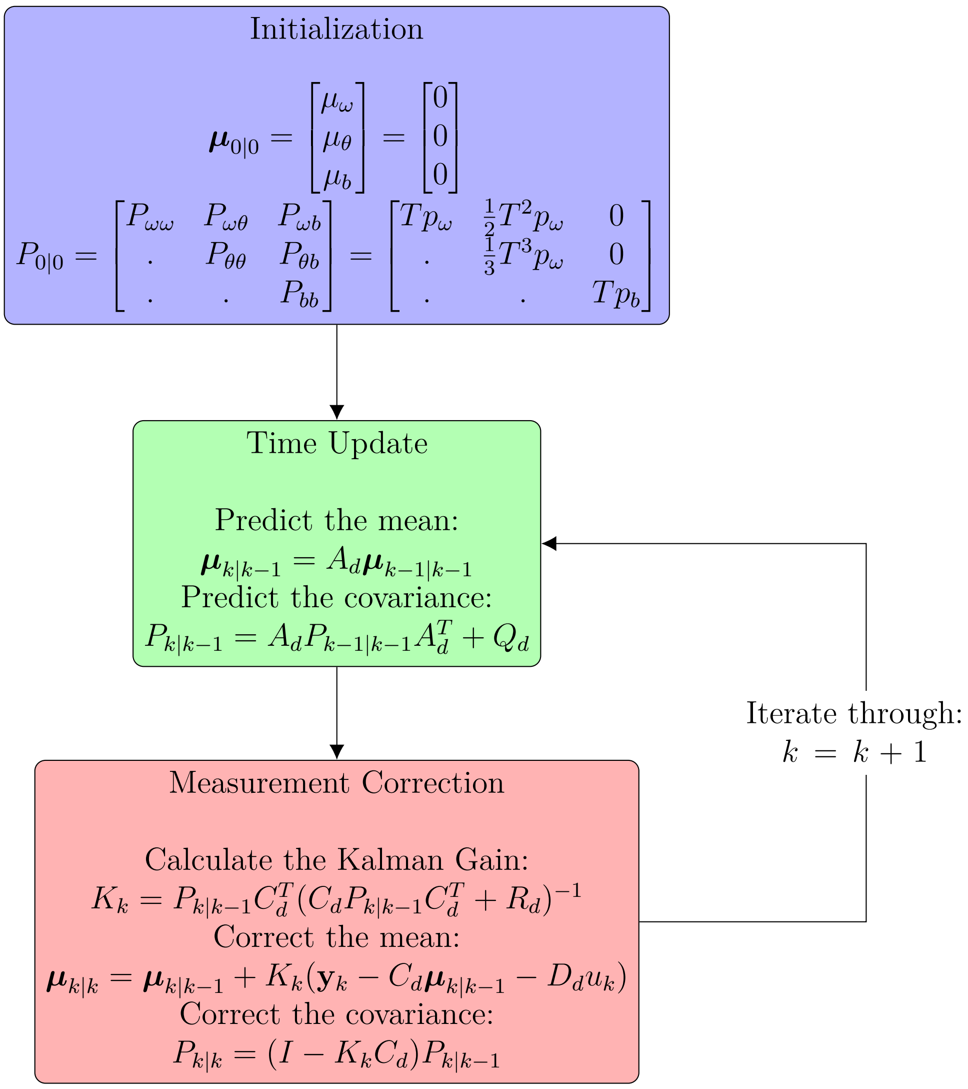
Initialization
$\boldsymbol{\mu}_{0|0} = \begin{bmatrix} \mu_\omega \\ \mu_\theta \\ \mu_b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
$P_{0|0} = \begin{bmatrix} P_{\omega\omega} & P_{\omega\theta} & P_{\omega b} \\ . & P_{\theta\theta} & P_{\theta b} \\ . & . & P_{bb} \end{bmatrix} = \begin{bmatrix} Tp_\omega & \frac{1}{2}T^2 p_\omega & 0 \\ . & \frac{1}{3}T^3 p_\omega & 0 \\ . & . & Tp_b \end{bmatrix}$
Time Update
Predict the mean:
$\boldsymbol{\mu}_{k|k-1} = A_d \boldsymbol{\mu}_{k-1|k-1}$
Predict the covariance:
$P_{k|k-1} = A_d P_{k-1|k-1} A_d^T + Q_d$
Iterate through:
$k = k + 1$
Measurement Correction
Calculate the Kalman Gain:
$K_k = P_{k|k-1} C_d^T (C_d P_{k|k-1} C_d^T + R_d)^{-1}$
Correct the mean:
$\boldsymbol{\mu}_{k|k} = \boldsymbol{\mu}_{k|k-1} + K_k(\mathbf{y}_k - C_d \boldsymbol{\mu}_{k|k-1} - D_d u_k)$
Correct the covariance:
$P_{k|k} = (I - K_k C_d) P_{k|k-1}$

Figure 2: Flow diagram of Kalman Filter showing prediction and correction of the mean and covariance.

Table 1: Kalman filter tuning parameters.

Tuning Constant	[units]	Description
p_ω	[rad$^2\cdot$ sec^{-2}]	Initial ω state uncertainty
p_b	[rad$^2\cdot$ sec^{-2}]	Initial b state uncertainty
q_ω	[rad$^2\cdot$ sec^{-2}]	ω state noise
q_b	[rad$^2\cdot$ sec^{-2}]	b state noise
r_θ	[rad$^2\cdot$ sec]	θ measurement noise
r_ω	[rad$^2\cdot$ sec^{-1}]	ω measurement noise

Task 3 – Estimating wheel angular velocity and angle from encoder readings

1. Use the code in Listing 2, or otherwise, to estimate the angular velocity $\dot{\varphi}$ and angle φ of the wheel using the encoder counts. *Note:* you have to change some constants in order to correct estimates.

2. Implement the estimation of the wheel angular velocity and angle separately for the left and right wheels.

Listing 2: Arudino code for using interrupts to count the number of encoder pulses and calculate the elapsed time between each pulse to estimate the angular velocity and angle of the wheel.

```
// Define the pins for the encoder
#define ENCODER_A 2
#define ENCODER_B 3
volatile uint8_t encoderALast;
volatile uint8_t encoderBLast;

// Define constants
const float WHEEL_RADIUS = 0.045;   // Radius of the wheel in meters
const float GEAR_RATIO = 50.0;   // Gear ratio of the encoder
const int ENCODER_PULSE_PER_REVOLUTION = 28;   // Number of encoder
    pulses per revolution
const float COUNTS_PER_REVOLUTION = ENCODER_PULSE_PER_REVOLUTION *
    GEAR_RATIO; // Pulses per revolution
// Define the variables for the encoder
volatile int16_t encoder_counts = 0;
volatile unsigned long last_time = 0;
volatile float position_change = 0.0;
volatile float angular_velocity = 0.0;
volatile float angle = 0.0;
void setup() {
  // Set the encoder pins as inputs and enable pull-up resistors
  pinMode(ENCODER_A, INPUT_PULLUP);
  pinMode(ENCODER_B, INPUT_PULLUP);
  // Attach interrupts to the encoder pins
  attachInterrupt(digitalPinToInterrupt(ENCODER_A), encoder_isr, CHANGE);
  attachInterrupt(digitalPinToInterrupt(ENCODER_B), encoder_isr, CHANGE);
  // Initialize serial communication
  Serial.begin(9600);
}

void loop() {
  // Calculate the elapsed time since the last encoder pulse
  unsigned long current_time = micros();
   // Calculate the time difference since the last interrupt
  float dt_interrupt = (float)(current_time - last_time) / 1000000;
  // Calculate the change in the position of the encoder in radians
  position_change = 2 * PI * encoder_counts / COUNTS_PER_REVOLUTION;
  // Reset encoder counts
  encoder_counts = 0;
  // Calculate the angular velocity
  angular_velocity = position_change / dt_interrupt;
  // Calculate the angle
  angle += position_change;
  // Print results
  Serial.print("Angular␣velocity:␣");
  Serial.print(angular_velocity);
  Serial.print("␣rad/s,␣Angle:␣");
  Serial.print(angle);
  Serial.println("␣radians");
  // Update last time
  last_time = current_time;
}

void encoder_isr() {
  uint8_t encoderAState = digitalRead(ENCODER_A);
  uint8_t encoderBState = digitalRead(ENCODER_B);

  if (encoderALast*encoderBLast) { //State 11
    encoder_counts -= (encoderAState - encoderBState);
  } else if (encoderALast && (encoderALast^encoderBLast)) { //State 10
    encoder_counts += encoderAState*encoderBState - (1-encoderAState|encoderBState);
  } else if (encoderBLast && (encoderBLast^encoderALast)) { //State 01
    encoder_counts += (1-encoderAState|encoderBState) - encoderAState*encoderBState;
  } else { //State 00
    encoder_counts += encoderAState - encoderBState;
  }

  encoderALast = encoderAState;
  encoderBLast = encoderBState;
}
```

Acknowledgments
This laboratory was developed by Roxanne R. Jackson, Magnus Axelson-Fisk, and Tom J. Jüstel. ChatGPT (AI-generated text) was used to assist in writing the content of this laboratory.