Home Exam

Henrik Söderlund (**hesa0077**)

September 18, 2019

Telerobotics and Applied Sensor Fusion, 7,5hp

1 Part A

1.1 Question 1

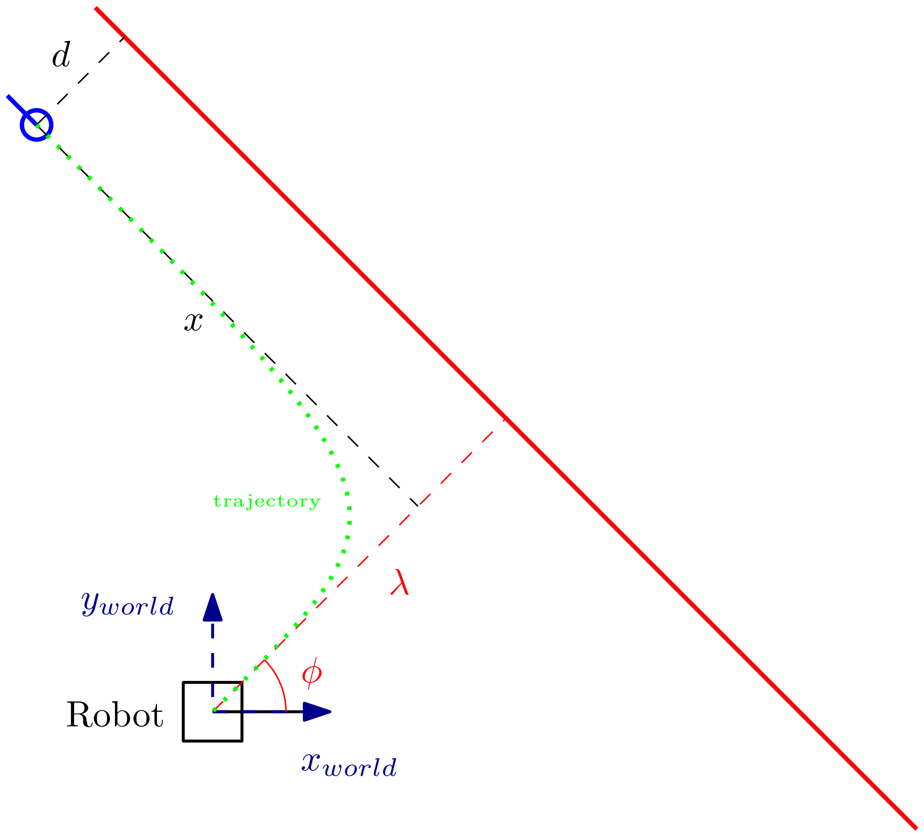

Figure 1 – Illustration of the task in Part A.

1.2 Question 2

We can extract the position and the vector of the Hough line from two Hough transform variables λ and ϕ, where λ is the distance to the Hough line and ϕ is the angle to the Hough line.

Firstly, we define a rotation matrix that describes the rotation of the vector to the Hough line as

$$R_\phi = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix},$$

where $v_{hough} = R_\phi\,[1,\ 0]^T$ is the direction vector to the Hough line.

Next, we obtain the Hough line origin by computing $p_{line} = \lambda\, v_{hough}$ and we compute the vector describing the direction of the Hough line as $v_{line} = R_{90^\circ}\, v_{hough}$, where R_{90° is

$$R_{90^\circ} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

1.3 Question 3

The linear velocity of a differential driven robot can be written as [1]

$$V_{robot} = \frac{R}{2}(\omega_r + \omega_l), \tag{1}$$

where ω_r is the angular velocity of the right wheel and ω_l is the angular velocity of the left wheel. R is the wheel radius.

The steering angle of a differential robot can be written as [1]

$$\omega_{robot} = \frac{R}{L}(\omega_r - \omega_l), \tag{2}$$

where L is the distance between the wheels on the drive axis.

The robot speed in the world frame is computed as [1]

$$\begin{cases} \dot{x} = V_{robot}\ \cos\theta \\ \dot{y} = V_{robot}\ \sin\theta \\ \dot{\theta} = \omega_{robot} \end{cases} \tag{3}$$

where θ is the heading angle of the robot in world frame.

We can now substitute (1) and (2) into (3) and write it in matrix form and in discrete time as [1]

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \Delta t \underbrace{\begin{bmatrix} \frac{R}{2}\cos\theta_k & \frac{R}{2}\cos\theta_k \\ \frac{R}{2}\sin\theta_k & \frac{R}{2}\sin\theta_k \\ \frac{R}{L} & -\frac{R}{L} \end{bmatrix}}_{M} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix}.$$

Moreover, if we have a controller that outputs a linear velocity v_c and steering angle γ_c as control signals, we can compute the applied wheel speeds by inverting M with the pseudo inverse such that

$$\begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} = M^* \begin{bmatrix} v_c \cos\theta \\ v_c \sin\theta \\ \gamma_c \end{bmatrix}, \quad M^* = (M^T M)^{-1} M^T.$$

1.4 Question 4

The first control law is based on [2]. Given a goal point such that we have distance to goal ρ, angle to goal α and angle to approach goal β, we can create a control law that ensures a certain pose is achieved of our mobile robot.

From our robot pose $\zeta_r = [x_r, y_r, \theta_r]^T$ and the goal pose $\zeta_g = [x_g, y_g, \theta_g]^T$, we can compute the controllable states as [2]

$$\begin{cases} \rho = \sqrt{\Delta x^2 + \Delta y^2} \\ \alpha = \tan^{-1}\frac{\Delta y}{\Delta x} - \theta_r \\ \beta = \theta_g - \theta_r - \alpha \end{cases}$$

where $\Delta x = x_g - x_r$ and $\Delta y = y_g - y_r$.

We update the states using the following discrete model [2]:

$$\begin{bmatrix} \rho_{k+1} \\ \alpha_{k+1} \\ \beta_{k+1} \end{bmatrix} = \begin{bmatrix} \rho_k \\ \alpha_k \\ \beta_k \end{bmatrix} + \Delta t \begin{bmatrix} -\cos\alpha_k & 0 \\ \frac{sin\alpha_k}{\rho_k} & -1 \\ -\frac{sin\alpha_k}{\rho_k} & 0 \end{bmatrix} \begin{bmatrix} v \\ \gamma \end{bmatrix}, \quad if\ \alpha \in \left(-\frac{\pi}{2},\ \frac{\pi}{2}\right]$$

and we assume the goal is in front of the vehicle (otherwise, we have to align the robot so it is pointing in the direction of the goal). The linear control law can be written as [2]

$$\begin{cases} v = k_\rho \rho \\ \gamma = k_\alpha \alpha + k_\beta \beta \end{cases}.$$

In order for the control law to be stable, the control gains has to fullfill the following inequalities: $k_\rho > 0,\ k_\beta < 0,\ k_\alpha - k_\rho > 0$ [2].

The control gains are chosen as $k_\rho = 0.3,\ k_\alpha = 2,\ k_\beta = -0.3$ in this case.

The second control law is made out of intuition. The goal with this controller is to try to reach the desired distance d from the wall as quickly as possible.

Firstly, we compute the projection of the robot onto the vector describing the direction from world origin to the wall. This can be written as

$$proj_{v_{hough}}(p_r) = p_r \cdot v_{hough}, \quad p_r = [x_r,\ y_r]^T,$$

where $\cdot$ is the dot product.

We can compute the relative distance $\Delta\lambda$ from the robot to the wall as

$$\Delta\lambda = \lambda - proj_{v_{hough}}(p_r).$$

The control signal γ is computed from the error $e = d - \Delta\lambda$ as

$$\begin{cases} \gamma = -k_\gamma e, \quad alpha < 0 \\ \gamma = k_\gamma e, \quad alpha > 0 \\ \gamma = 0, \quad alpha = 0 \end{cases},$$

where the control gain k_γ is chosen as $k_\gamma = 0.1$.

Let the control signals of the first control law be denoted by the subscript $(\cdot)_1$ and the second control law $(\cdot)_2$. The resulting control signals from both control laws are:

$$\begin{cases} v_{total} = v_1 \\ \gamma_{total} = \gamma_1 + \gamma_2 \end{cases}.$$

1.5 Question 5

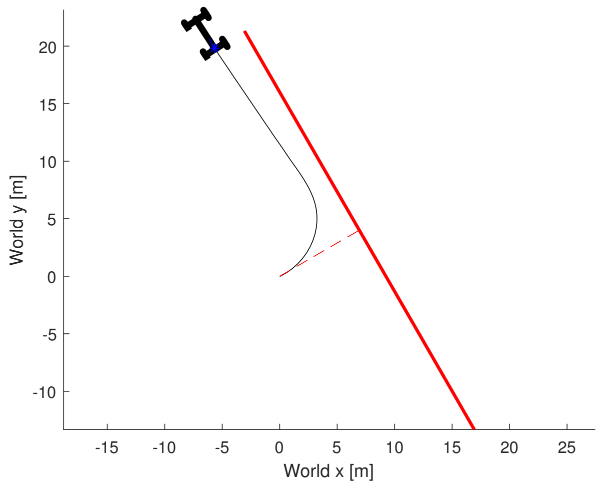

Figure 2 – Trajectory of the robot in Part A.

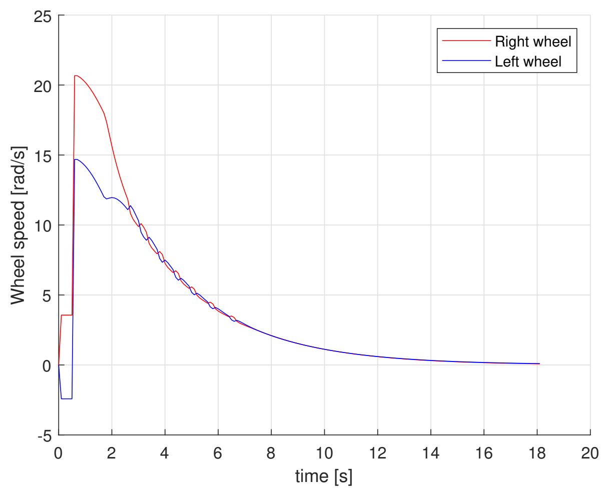

Figure 3 – Applied wheel angular velocities in Part A.

1.6 Question 6

For QuadDrive(20,3), see Figures 2-3.

The figures below are for QuadDrive(-20,3):

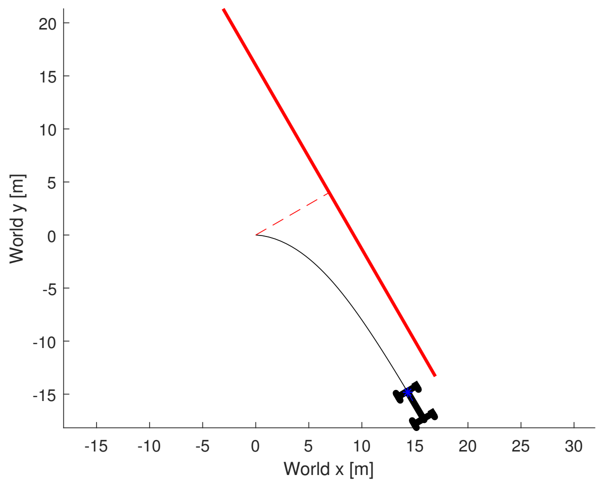

Figure 4 – Trajectory of the robot in Part A, using QuadDrive(-20,3).

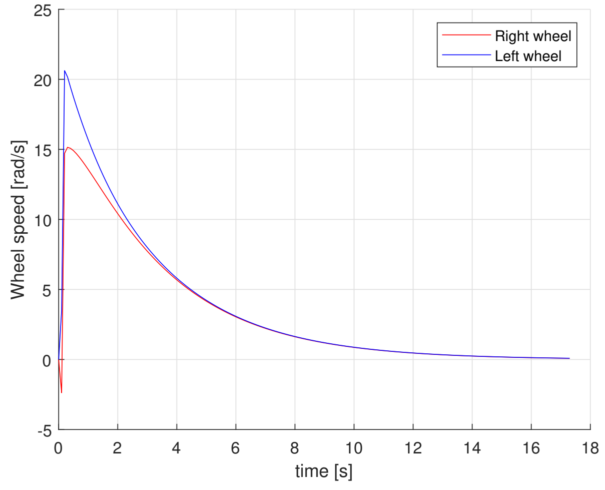

Figure 5 – Applied wheel angular velocities in Part A, using QuadDrive(-20,3).

1.7 Question 7

A telecommand "QuadDrive" is used in this task. When the simulation runs, it prompts the user for a telecommand:

```
Enter telecommand:
```

Say that the user types

```
Enter telecommand: QuadDrive
```

then, the simulator prompts for the x and d variables:

```
Enter distance to drive along wall: 20

Enter distance from wall: 3
```

After the variables are defined, the simulation executes the telecommand. When the telecommand is complete, the simulator prompts for a new command. The user can then choose to continue driving along the wall with additional "QuadDrive" commands.

Another useful command would be to choose a wall to track with "ChooseWall" and then the user gets a list of trackable walls that can then be used with "QuadDrive". A command "BackToStart" to navigate back to the starting pose could be useful also.

I also made a telecommand for exiting the simulation called "exit".

2 Part B

2.1 Question 1

Part A has perfect knowledge of the environment and thus knows the exact pose of the Hough line in world frame. Part B does not know where the wall is, but has to instead rely on a lidar sensor mounted somewhere on the robot. The sensor is quite bad, with gaussian distributed noise $\mathcal{N}(0, 0.1)$ for range and $\mathcal{N}(0, 7^\circ)$ for angle.

2.2 Question 2

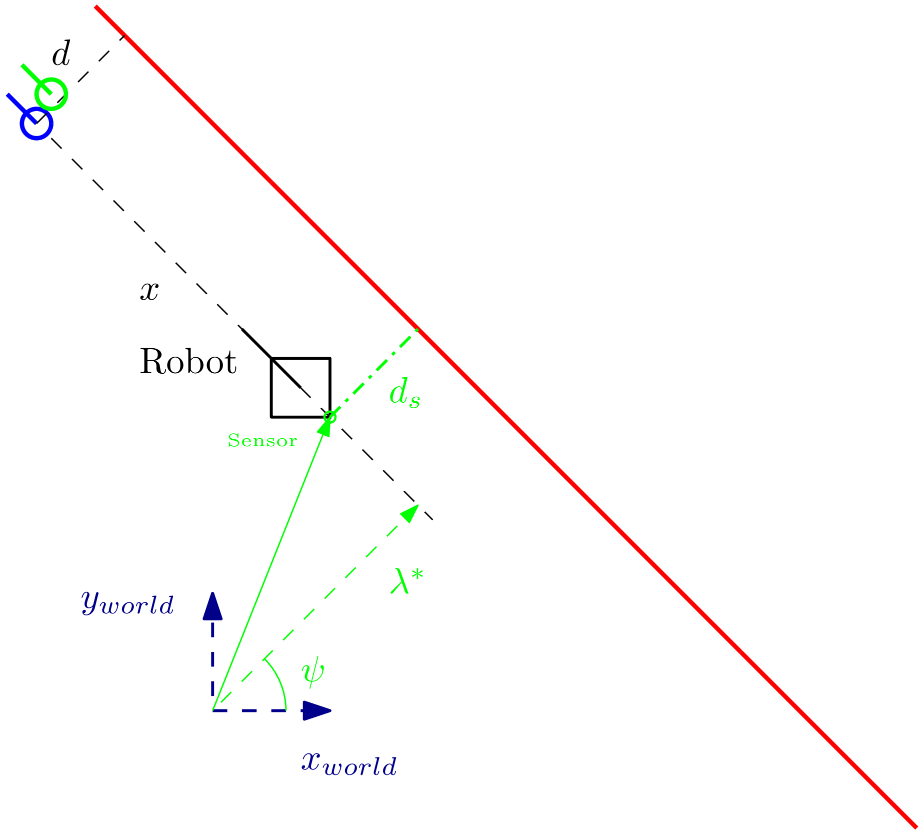

Figure 6 – Illustration of the task in Part B.

2.3 Question 3

The sensor is mounted on the center of the rear wheel axis, as can be seen in Figure 6. The relative position of the sensor is thus $p_s^r = [0,\ 0]^T$. This provides better stability since there will be less oscillations in the laser sensor reading because it will move much slower in relation to the steering.

2.4 Question 4

Sensor fusion can be implemented with a simple complementary filter. By taking into consideration the previous measurement x, we can choose how much to update with the observation z. Since we will use the same controller as before, we compute the goal pose by projecting the sensor position onto a vector with direction ψ and magnitude λ^*, where ψ is the angle from the sensor and λ^* is the scalar projection from the sensor position onto the vector with direction ψ. We can then use the obtained sensor range reading d_s to compute $\lambda = \lambda^* + d_s$ and we now have a vector pointing from the world origin to the Hough line.

The complementary filter is defined as

$$x_{k+1} = (1 - \alpha)x_k + \alpha z_k,$$

where we choose $\alpha = 0.1$.

I was thinking of using a method with covariance matrices, but couldn't think of a way of using it without a second sensor. I could probably use the "perfect" navigation system on the robot but that didn't feel right. I do compute the covariance matrix though and I do plot it onto the estimated goal position.

2.5 Question 5

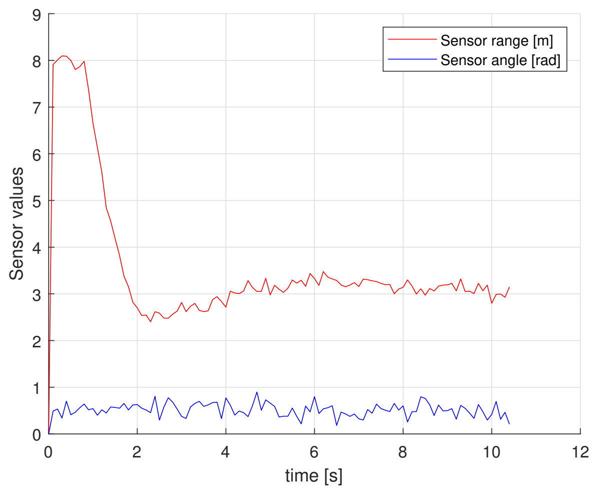

Figure 7 – Sensor readings in Part B.

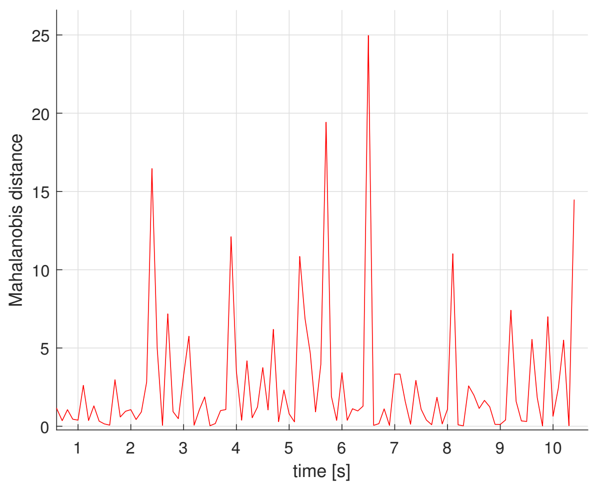

Figure 8 – Mahalanobis distance in Part B.

2.6 Question 6

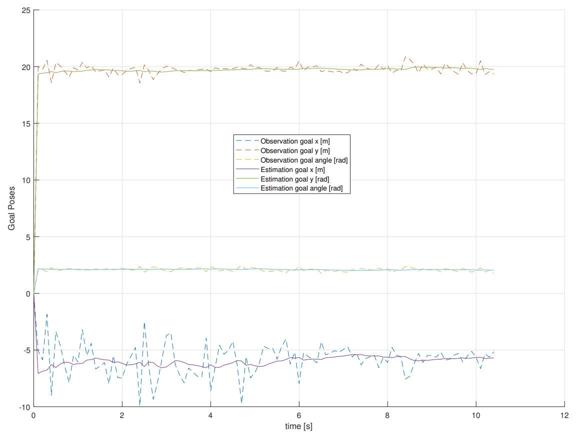

Figure 9 – Goal estimations and observations in Part B.

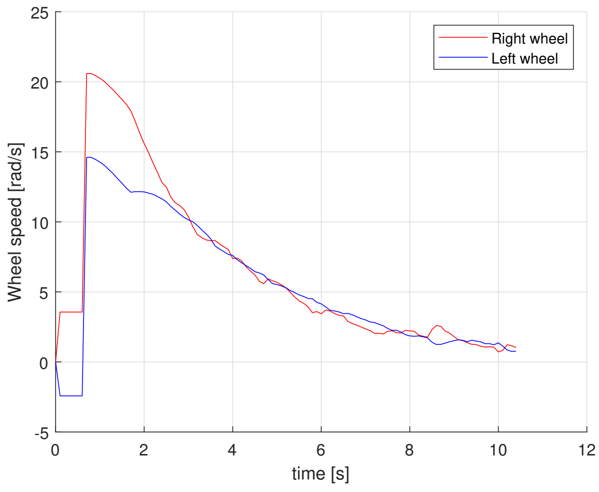

Figure 10 – Applied wheel angular velocities in Part B.

2.7 Question 7

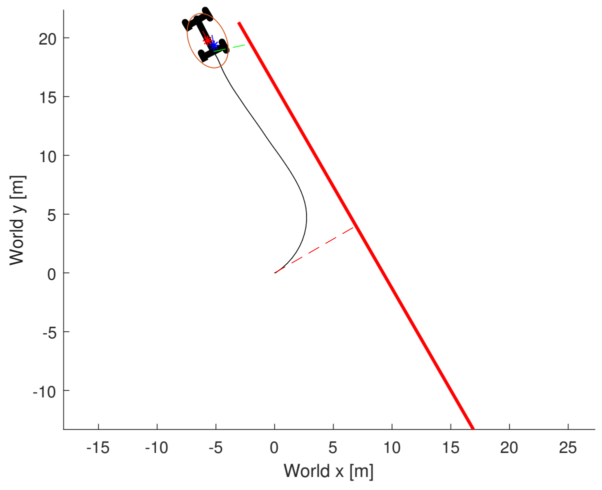

Figure 11 – Trajectory in Part B.

2.8 Question 8

I already did this... Figure 7 for observations, Figure 10 for control signals.

3 Part C

3.1 Question 1

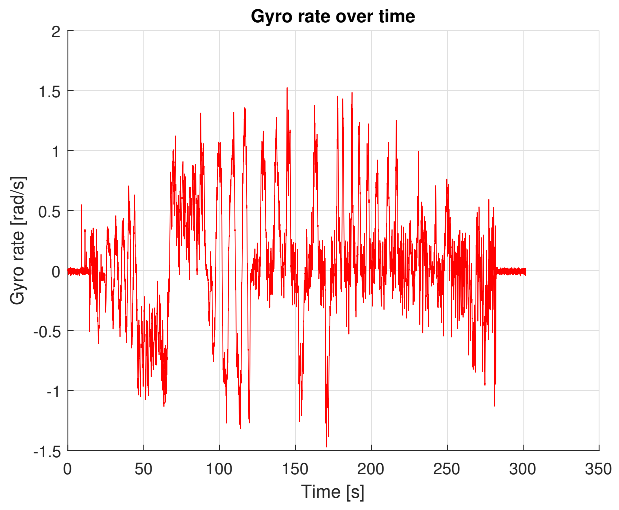

Figure 12 – Gyro rate in Part C.

3.2 Question 2

A bias is an inclination towards a certain direction, which in terms of sensors means that the sensor value tends to lean towards a certain offset. The physical properties of MEMS gyroscopes change over time, which means that the bias is also time dependent.

The bias of a gyroscope is modelled as a brownian motion, or "random walk". This is because the bias will vary over time due to things like flicker noise in the electronics [3].

3.3 Question 3

A good place to measure the bias point in the gyro data seems to be between 280 and 300 seconds, as can be seen in Figure 13. This is because there is not much movement here so the dominant signal is the white noise.

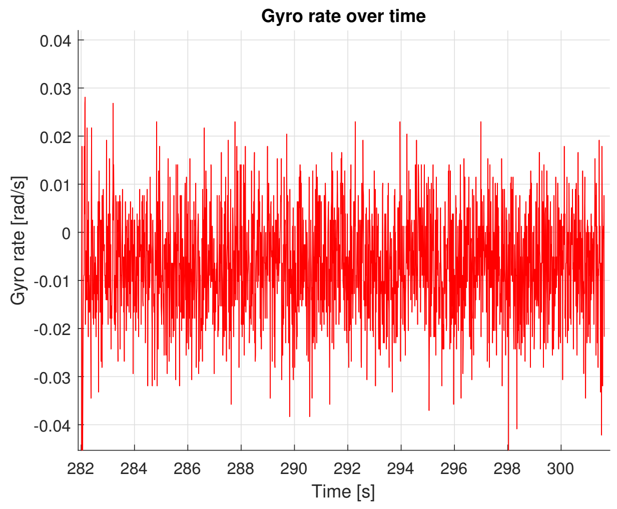

Figure 13 – Good place to measure bias in Part C.

3.4 Question 4

Let the gyro rate be called $\omega_g(t)$, most gyroscopes are modelled as

$$\tilde{\omega}_g(t) = \omega_g(t) + \omega_b(t) + n(t),$$

where $\omega_b(t)$ is the bias point and $n(t)$ is white noise.

We have to assume that the input U will be zero, since we will use the gyro rate as observations, thus we do not need to compute the input vector G. The state vector is chosen as:

$$x_k = \begin{bmatrix} \theta_k \\ (b_g)_k \end{bmatrix},$$

where b_g is the estimated bias and θ is the estimated angle. The state transition matrix F is chosen as:

$$F_k = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}.$$

The observation vector is chosen as

$$z_k = \tilde{\omega}_g(t).$$

The observation model is chosen as

$$H = [1,\ 0].$$

3.5 Question 5

The integrated signal within the chosen interval looks like the following:

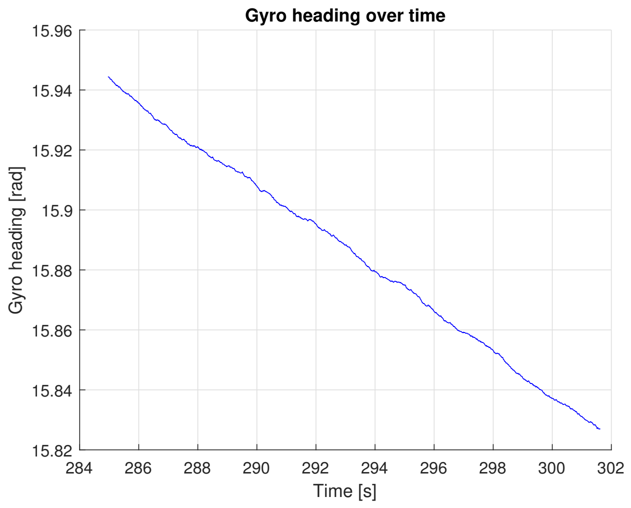

Figure 14 – Integrated signal within the chosen interval in Part C.

The estimated bias from the kalman filter developed as the following:

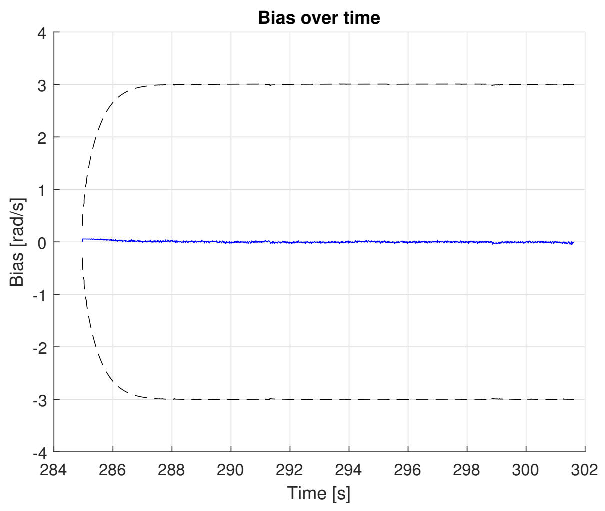

Figure 15 – Estimated bias within the chosen interval in Part C.

The innovation and the confidence intervals resulted in:

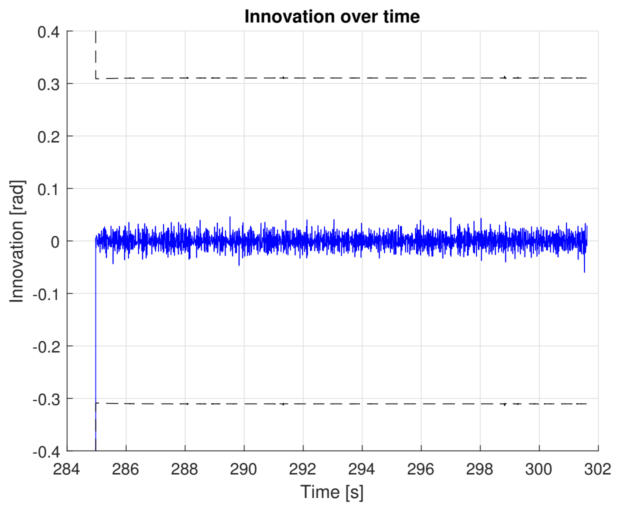

Figure 16 – Innovation within the chosen interval in Part C.

Here I used the following noise covariance matrices (R is a 1×1 matrix):

$$Q = \begin{bmatrix} 0.1^2 & 0 \\ 0 & 0.1^2 \end{bmatrix},$$

$$R = \left(\frac{\pi}{180}\right)^2.$$

The predicted bias ended up as $\omega_b = 0.0076$.

3.6 Question 6

A Kalman filter can be trimmed by adjusting the process noise matrix Q and the measurement noise matrix R. The process noise matrix determines how fast the confidence intervals will change as the innovation increases. The measurement noise matrix determines the bar of the innovation which is acceptable noise and does not have to be adjusted for.

3.7 Question 7

A validation gate is a condition under which an observation is deemed acceptable. Often the validation gate comprises a statistical measure for determining if the observation is correlated with the measurement or not. A common measure is the Mahalonobis distance

$$d_k^2 = (z_k - H_k\hat{x}_{k|k-1})^T S_k^{-1}(z_k - H_k\hat{x}_{k|k-1}),$$

which is χ^2 distributed and unit-less. This means that we can use the confidence intervals provided in the χ^2-table for a certain DOF to use as the validation gate.

3.8 Question 8

I implemented the validation gate using Mahalonobis distance. I check if d_k^2 is within 95% of the confidence interval with 1 DOF (since we have a one dimensional observation), i.e. $d_k^2 <= 3.841$. If the condition is satisfied, the new estimation is computed with the Kalman gain, otherwise the estimation is equal to the prediction and the Kalman gain is zero.

3.9 Question 9

I trimmed the noise covariance matrices so that better performance was achieved. The new matrices are:

$$Q = \begin{bmatrix} 0.003^2 & 0 \\ 0 & 0.03^2 \end{bmatrix},$$

$$R = \left(\frac{0.1\pi}{180}\right)^2.$$

The trimmed estimated bias from the kalman filter developed as the following:

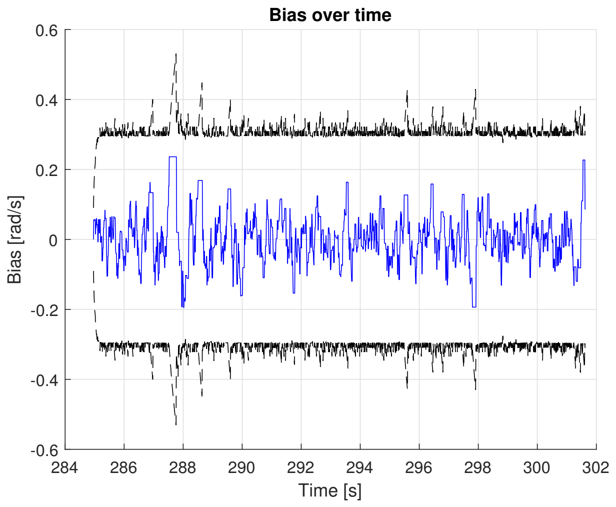

Figure 17 – Trimmed estimated bias within the chosen interval in Part C.

The trimmed innovation and the confidence intervals resulted in:

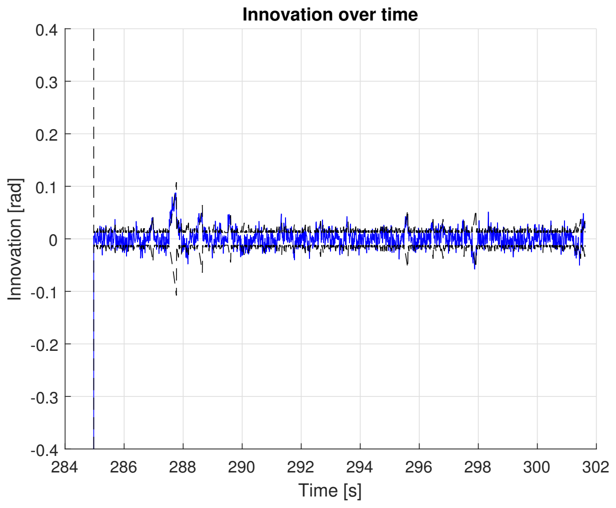

Figure 18 – Trimmed innovation within the chosen interval in Part C.

The predicted bias ended up as $\omega_b = 0.0855$.

3.10 Question 10

See Figure 18 for plot of innovation.

The innovation vector is the difference between the current observation and the predicted observation, meaning that if the measurement is equal to the observation, the innovation is zero. Optimally, the innovation should be just white noise within the confidence intervals computed by the Kalman filter, since then the filter successfully predicts the observation.

The innovation can be used to determine how good an observation is, together with the validation gate.

3.11 Question 11

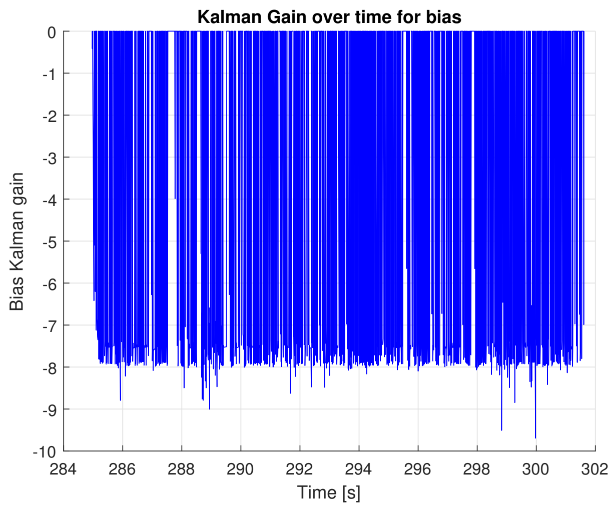

Figure 19 – Bias Kalman gain in Part C.

The Kalman gain determines how much of the observation should be used to compute the estimated states. The higher the Kalman gain, the more of the observation will be used, the lower the Kalman gain, the more the predicted measurement will be used.

3.12 Question 12

See Figure 17 for b_g and the figure below for θ:

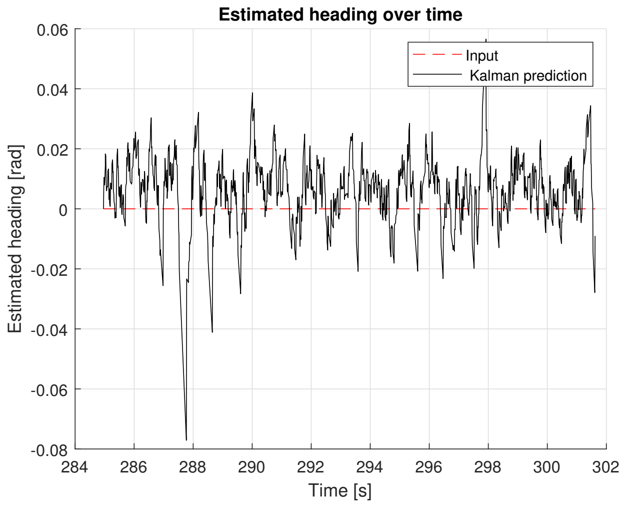

Figure 20 – Estimated heading in Part C.

3.13 Question 13

We did this in lab 3.

References

[1] Hirpo et al., Design and Control for Differential Drive Mobile Robot, International Journal of Engineering Research & Technology (IJERT), 6(10), pp. 327–334, 2017.

[2] Corke, Peter. Robotics, Vision and Control : Fundamental Algorithms in MATLAB. Springer, 2011.

[3] www.vectornav.com, Gyroscope, accessed 18-01-2019.