EE4308 Autonomous Robot Systems

AY2021/2022 Semester 2

Project 2

Team 19

Philippe Brigger, Friedrich Ginnold
Maria Krinner, Philip Wiese

Contents

.

1 Introduction

This report is part of the module *EE4308 Autonomous Robot Systems* at the National University of Singapore (NUS) in spring 2022.

In summary, the following modifications of the provided code for the hector drone have been made:

1. Finite state machine (Section 2)
2. Three dimensional trajectory generation with
   (a) Cubic Hermite splines (Section 2.1)
   (b) Look-ahead time (Section 2.1)
   (c) Prediction of turtle motion (Section 2.2)
3. PID motion controller (Section 3)
4. Extended Kalman filter (EKF) (Section 4) for pose estimation using
   (a) Inertial Measurement Unit including bias calibration (Sections 4.1 and 5.3)
   (b) Global Positioning System (Section 4.2.1)
   (c) Magnetometer (Section 4.2.2)
   (d) Barometer including bias estimation (Sections 4.2.3 and 5.1)
   (e) Sonar (Section 4.2.4)

In simulation, the hector drone takes off and flies to a height of 2 m. The project task was to make the drone continuously fly from the starting point to the moving turtle ground robot, from the turtle to the turtle's final goal and from the final goal back to hector's starting position. During this cycle, the drone is constantly yawing at 0.5 rad/s and does not exceed a maximum linear speed of 2 m/s in the xy-plane respectively 0.5 m/s along z. Additionally, the drone must come within 0.2 m of a waypoint before flying to its next target.

Detailed implementation details of the EKF are provided in Section 4.3. Further, all simulations results are presented in Section 5. A detailed plot including the estimated values, error compared to the true values and the influence of the various sensor on the correction is presented in Section 5.4. Finally, a short conclusion including the key learnings and possible future work is given in Section 6.

2 Finite State Machine

The diagram for the finite state machine is shown in Figure 1. During each iteration of the main process, the coordinate goal is updated according to the current state. As soon as the distance between the drone and the goal is smaller than the configurable `close_enough` distance, a state transition is executed. When the turtle reaches its final goal, the drone completes the current cycle, and then lands.

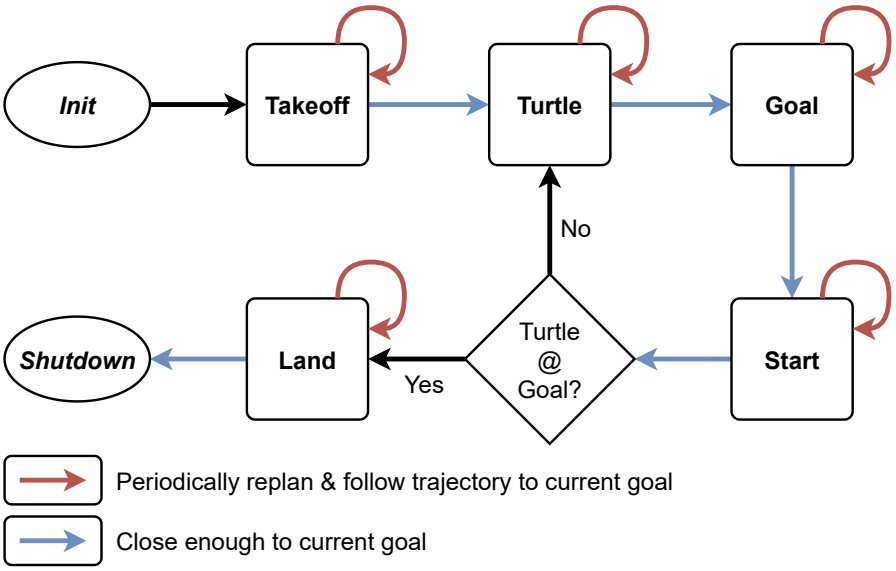


Figure 1: Finite State Machine of the Hector Drone.

2.1 Trajectory Generation and Look-Ahead Distance

During flight, intermediate targets with a distance defined by the `look_ahead` parameter are generated as shown in Figure 2. Note that this parameter is configurable in the `hector.yaml` file.

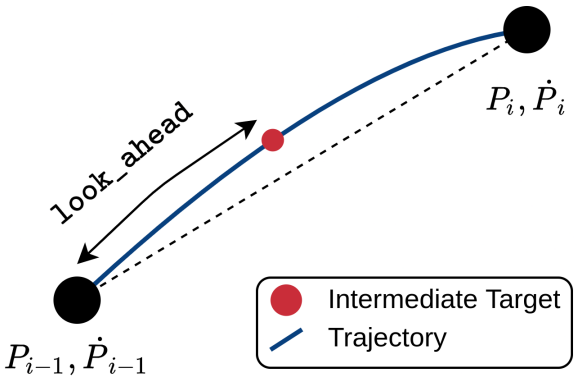


Figure 2: Illustration of the `look_ahead` distance. P_{i-1} is the current drone position and P_i the goal.

The trajectory is regenerated and a new intermediate target is calculated if one of the following conditions is met:

- The distance between the drone and the intermediate target is smaller than `look_ahead` − `close_enough`
- The distance between the drone and the goal is smaller than `close_enough`

The trajectories are generated with three-dimensional cubic Hermite splines. Hereby, the same approach used for *Project 1* was extended to three dimensions. For the sake of completeness, the underlying equations are listed again. Following the lecture notes, the trajectory can be calculated with

$$\begin{aligned} x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \dot{x}(t) &= a_1 + 2a_2 t + 3a_3 \\ y(t) &= b_0 + b_1 t + b_2 t^2 + b_3 t^3 \\ \dot{y}(t) &= b_1 + 2b_2 t + 3b_3 \\ z(t) &= c_0 + c_1 t + c_2 t^2 + c_3 t^3 \\ \dot{z}(t) &= c_1 + 2c_2 t + 3c_3 \end{aligned}$$

The parameters of the cubic Hermite spline are calculated with

$$\begin{aligned} \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \end{bmatrix}^\mathsf{T} &= \mathbf{M}^{-1} \begin{bmatrix} x_{i-1} & \dot{x}_{i-1} & x_i & \dot{x}_i \end{bmatrix}^\mathsf{T} \\ \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix}^\mathsf{T} &= \mathbf{M}^{-1} \begin{bmatrix} y_{i-1} & \dot{y}_{i-1} & y_i & \dot{y}_i \end{bmatrix}^\mathsf{T} \\ \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \end{bmatrix}^\mathsf{T} &= \mathbf{M}^{-1} \begin{bmatrix} z_{i-1} & \dot{z}_{i-1} & z_i & \dot{z}_i \end{bmatrix}^\mathsf{T} \end{aligned}$$

with $\mathbf{M}^{-1}$ defined by

$$\mathbf{M}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{\Delta t^2} & -\frac{2}{\Delta t} & \frac{3}{\Delta t^2} & -\frac{1}{\Delta t} \\ \frac{2}{\Delta t^3} & \frac{1}{\Delta t^2} & -\frac{2}{\Delta t^3} & \frac{1}{\Delta t^2} \end{bmatrix}$$

where Δt denotes the flight time between P_{i-1} and P_i. For the first point P_{i-1}, the current vehicle velocity is used. The velocity at the goal point P_i is arbitrarily set to zero.

To visualize the trajectory in RViz and assuming the drone is flying with its average speed v_{avg}, temporal equidistant points are published to the `trajectory` topic.

2.2 Turtle Position Estimation

In order to estimate the future turtle position, the flight time between the current drone position P_{i-1} and the current turtle position P_i using the drone's average speed is estimated.

$$\Delta t = \frac{||P_{i-1} - P_i||}{v_{avg}}$$

Further, the velocity of the turtle is needed to estimate its future position after the time Δt.

The first approach is to directly use the turtle velocity available from the motion filter running on the turtle node. However, if we want to reduce the communication overhead, the turtle velocity can also be estimated from the already available position information using the positions at t_k and t_{k-1}.

$$\begin{aligned} v_x &\approx \frac{x_k - x_{k-1}}{t_k - t_{k-1}} \\ v_y &\approx \frac{y_k - y_{k-1}}{t_k - t_{k-1}} \end{aligned}$$

Because this estimation results in poor performance due to noise, an additional infinite impulse response (IIR) filter in the form of a low pass is added.

$$\hat{v}_{x,k} = \alpha\, v_{x,k} + (1-\alpha)\hat{v}_{x,k-1}$$
$$\hat{v}_{y,k} = \alpha\, v_{y,k} + (1-\alpha)\hat{v}_{y,k-1}$$

Consequently, the estimated future turtle position $\hat{P}_i$ on time of arrival can be approximated according to

$$\hat{P}_{i,x} = P_{i,x} + \Delta t \cdot \hat{v}_{x,k}$$
$$\hat{P}_{i,y} = P_{i,y} + \Delta t \cdot \hat{v}_{y,k}$$

2.3 Evaluation

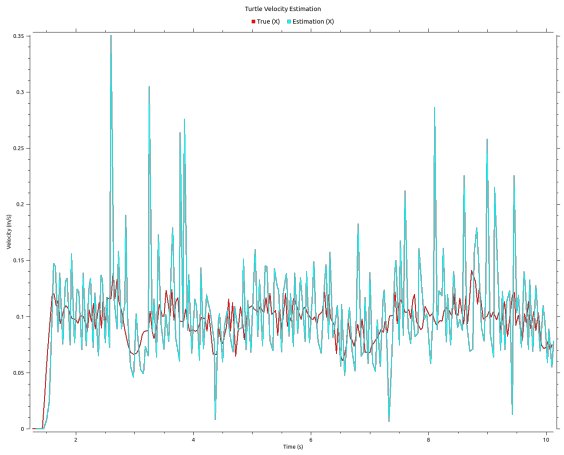
The final parameters, defined in `hector.yaml` and used for the finite state machine, trajectory generation and turtle position estimation, are shown in Table 1.

Table 1: Final parameters related to the trajectory generation and turtle position estimation for the hector drone.

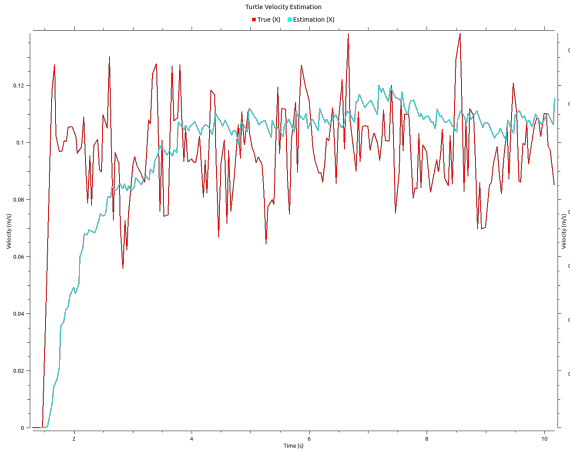
Parameter	**Value**	**Description**
`look_ahead`	1.0 m	Look-ahead distance for target generation
`weight_turtle_v`	0.5	Weight of low pass filter for turtle velocity

In order to determine a suitable value for `weight_turtle_v`, the true velocity of the turtle was compared to the velocity that the hector drone estimated. According to the results shown in Figure 3a, a too high value for α respectively `weight_turtle_v` does not dampen the noise well. On the other hand, a too low value results in a very slow response and could therefore significantly reduce the performance of the prediction, as shown in Figure 3c. Hence, an intermediate value $\alpha = 0.05$ was chosen. This value significantly reduces the effects of the noise while still providing reasonable response time, as seen in Figure 3b.

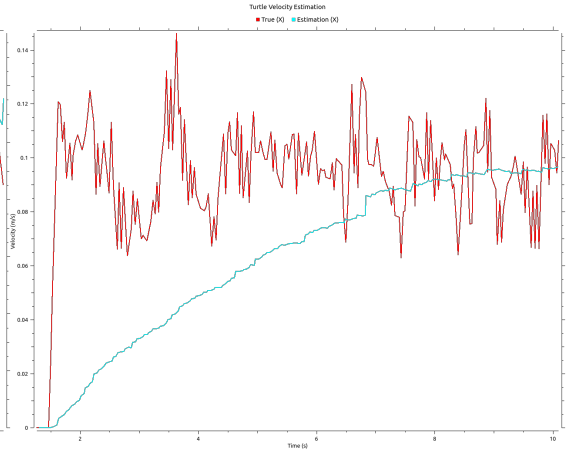
To determine a suitable value for the `look_ahead` distance, the following factors were considered. Because the state estimation is not perfect, some noise in the estimation is present. If the `look_ahead` distance is too small, the motion controller mainly reacts to this noise and the flight performance is decreasing considerably. Further, with a small value, the velocity of the drone is rather small, and it takes a lot of time to reach the goals. However, increasing the gains of the controller to increase the velocity leads to unstable behavior. On the other side, a too large value cancels the benefits of using spline trajectories to improve the continuity of the velocity. Hence, after multiple simulations, a value of 1 m was chosen, leading to a satisfying flight performance.



(a) $\alpha = 0.90$



(b) $\alpha = 0.05$



(c) $\alpha = 0.01$

Figure 3: IIR filter responses for different weight parameters for the turtle velocity estimation. The true value is shown in red and the estimation in blue.

3 PID Motion Controller

Similar to the first project, PID controllers are used to calculate the commanded velocities in the x, y and z-direction. The x, y and z-directions each use an independent controller, although the x and y controllers share the PID gains.

3.1 Commanded Velocity in z-Direction

Given the drone's current position (x_k, y_k, z_k) and the drone's target position (x_p, y_p, z_p), the error which is fed into the PID controller of the z direction is calculated as $\epsilon_z = z_p - z_k$. The commanded velocity in z-direction u_z is calculated as in project 1:

$$u_z = P_z + I_z + D_z \tag{1}$$

Note that there is a maximal velocity which is taken into account. When the velocity u_z is higher than that maximum, the controller commands the maximal velocity. The drone never has angular movement along its x or y-axis (see Figure 6 for the different coordinate frames). Therefore, the robot z-axis is always aligned with the world frame z-axis.

3.2 Commanded Velocity in the x-y Plane

This project requires the drone to move with a constant rotation around the z-axis. The commanded yaw rate is constant and defined in `hector.yaml`.

Similar to the calculation of the commanded velocity in the z-direction, the errors in the x-direction ϵ_x and y-direction ϵ_y from the drone's current position to the drone's target position are used to calculate the commanded drone velocities in world coordinates:

$$\hat{u}_x = P_x + I_x + D_x \tag{2}$$
$$\hat{u}_y = P_y + I_y + D_y \tag{3}$$

Because the position of the drone and the target are given in world coordinates, the calculated velocities $\hat{u}_x$ and $\hat{u}_y$ are also in world coordinates. The commanded velocities, however, are expected to be given as the drone's velocities in the robot frame. Because of the constant rotation around the drone's z-axis, the robot and world frame do not always match. Hence, a coordinate transformation of $\hat{u}_x$ and $\hat{u}_y$ needs to be carried out to get the final u_x and u_y in the robot frame. The transformation is done by using the yaw angle ψ of the hector:

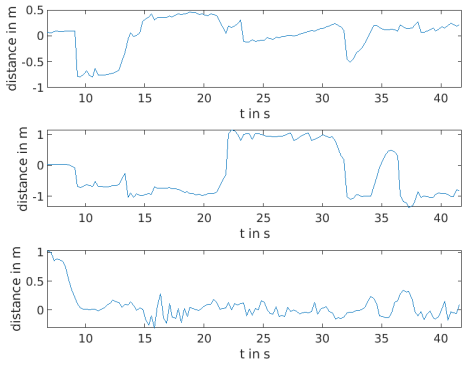
$$u_x = \hat{u}_x \cdot cos(-\psi) - \hat{u}_y \cdot sin(-\psi) \tag{4}$$
$$u_y = \hat{u}_x \cdot sin(-\psi) + \hat{u}_y \cdot cos(-\psi) \tag{5}$$

As for the velocity in the z-direction, there is a maximal velocity for the x and y-direction which is taken into account.

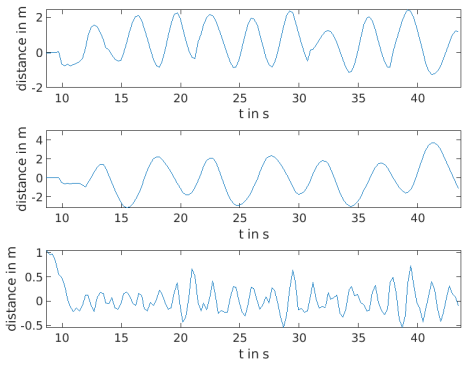
3.3 Tuning of the PID Controller Gains

The PID controllers have six parameters which need to be tuned. When the drone flies at a constant altitude, the target is always moving and there is no steady state. Therefore, it makes sense to set $K_{I,xy} = 0$. Since the target is incremented in small, discrete steps, the derivative part would become unstable due to the high derivatives appearing at each step increment. As a result, $K_{D,xy} = K_{D,z} = 0$ were chosen. Finally, the $K_{P,xy}$, $K_{P,z}$ and $K_{I,z}$ parameters were tuned iteratively.

A good solution was obtained with $K_{P,xy} = 1$, $K_{P,z} = 1$ and $K_{I,z} = 0.1$. When the proportional gains were set to high values, the drone occasionally overshot an intermediate target and oscillated



(a) $K_{P,xy} = 1$, $K_{P,z} = 1$, $K_{I,z} = 0.1$

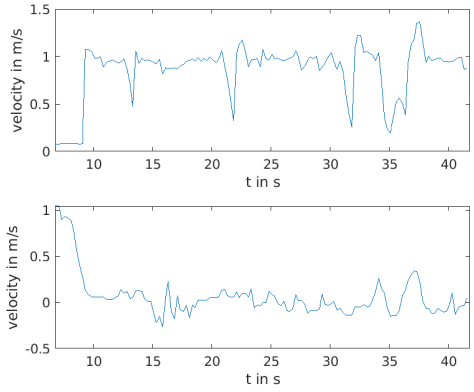


(b) $K_{P,xy} = 5$, $K_{P,z} = 5$, $K_{I,z} = 0.1$.

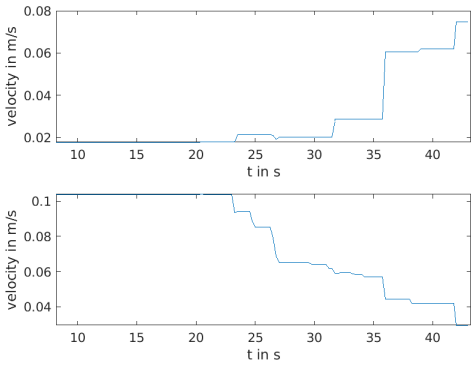
Figure 4: Distance between drone and target in x, y and z-direction in world coordinates for different sets of PID gains $K_{P,xy}$, $K_{P,z}$ and $K_{I,z}$. First row: Distance in x-direction. Second row: Distance in y-direction. Third row: Distance in z-direction.

around it before reaching it and continuing. This behavior was observed in all directions as shown in the error plots in Figure 4b and is undesirable.

Lower PID gains are in general more stable, as they make the robot follow the target without overshooting. However, the drone responds very slow, as seen in Figure 5b. Therefore, the PID gains should be chosen such that there is not much overshoot but the response remains quick.



(a) $K_{P,xy} = 1$, $K_{P,z} = 1$, $K_{I,z} = 0.1$



(b) $K_{P,xy} = 0.1$, $K_{P,z} = 0.1$, $K_{I,z} = 0.1$.

Figure 5: Drone velocity in xy-plane and z-direction in the robot frame for different sets of PID gains $K_{xy,lin}$, $K_{z,lin}$ and $K_{z,d}$. First row: Velocity in xy-plane. Second row: Velocity in z-direction.

4 Extended Kalman Filter

An extended Kalman filter (EKF) estimates the states of the hector drone. It is assumed that the roll and pitch angles of the hector are negligible. This strongly simplifies the calculations and filter implementation, as it may be assumed that the z-axis of the hector's robot frame is constantly aligned with the z-axis of the world frame. Therefore, the EKF does not have to estimate the roll and pitch angles, and the other states may be estimated independently. The EKF provides an estimate for the following states in the global world frame:

- x, y and z-position
- $\dot{x}$, $\dot{y}$ and $\dot{z}$ velocities
- Yaw angle ψ
- Angular velocity $\dot{\psi}$

An inertial measurement unit (IMU) predicts the positions and velocities. A global positioning system (GPS) provides additional position information of hector, while the magnetometer (MGN) measures the yaw angle. Finally, the barometer (BAR) along with the sonar (SNR) sensor are used to measure the drone's height. The measurements of the different sensors are fused in the EKF to correct the state prediction in order to provide robust and precise state estimates.

4.1 Inertial Measurement Unit

An inertial measurement unit is capable of measuring the angular velocities and linear accelerations of a system. The yaw angle ψ_k can be estimated by integrating the angular velocity measurement in the world frame v_ψ. Alternatively, the positions x, y, z and velocities $\dot{x}, \dot{y}, \dot{z}$ are found by integrating the linear acceleration measurements in the world frame $a_{x,k}$, $a_{y,k}$, $a_{z,k}$ once respectively twice.

$$\begin{bmatrix} x \\ y \\ z \\ \psi \end{bmatrix} = \begin{bmatrix} x_0 + v_x t + \frac{1}{2} a_x t^2 \\ y_0 + v_y t + \frac{1}{2} a_y t^2 \\ z_0 + v_z t + \frac{1}{2} a_z t^2 \\ v_\psi t \end{bmatrix} \tag{6}$$

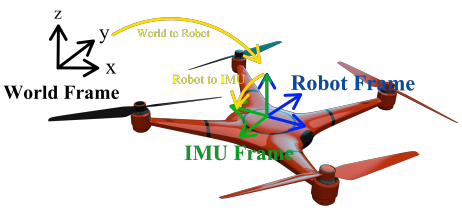


Figure 6: World, robot and IMU frame. The IMU measurements must be rotated from the IMU to the world frame.

It is to be noted that the IMU frame is rotated by 180 ° around the z-axis of the robot frame (see fig. 6). Due to the assumption of having negligible roll and pitch angles, the world frame linear accelerations a_x, a_y, a_z and world frame angular velocity v_ψ can be expressed with the

IMU measurements u_x, u_y, u_z, u_ψ by rotating the latter into the world frame:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \\ v_\psi \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_x \\ u_y \\ u_z - G \\ u_\psi \end{bmatrix} \tag{7}$$

The integrals are solved discretely, hence, Δt is the time step between two measurements, and $u_{x,k}$, $u_{y,k}$, $u_{z,k}$, $u_{\psi,k}$ are the linear accelerations respectively angular velocity of the robot measured by the IMU in the IMU frame at timestep k. Note that, $G = 9.8\,\mathrm{m\,s^{-2}}$ is the average gravity acceleration of earth.

The calculations of the positions, velocities, yaw angle and angular velocity are derived by inserting Equation (7) in Equation (6) and rewriting the equations in discrete time.

$$\begin{bmatrix} x_{k|k-1} \\ \dot{x}_{k|k-1} \end{bmatrix} = \begin{bmatrix} x_{k-1|k-1} + \dot{x}_{k-1|k-1}\Delta t + \frac{1}{2}(\Delta t)^2(-u_{x,k}\cos\psi_{k|k-1} + u_{y,k}\sin\psi_{k|k-1}) \\ \dot{x}_{k-1|k-1} + \Delta t(-u_{x,k}\cos\psi_{k|k-1} + u_{y,k}\sin\psi_{k|k-1}) \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} y_{k|k-1} \\ \dot{y}_{k|k-1} \end{bmatrix} = \begin{bmatrix} y_{k-1|k-1} + \dot{y}_{k-1|k-1}\Delta t - \frac{1}{2}(\Delta t)^2(u_{x,k}\sin\psi_{k|k-1} + u_{y,k}\cos\psi_{k|k-1}) \\ \dot{y}_{k-1|k-1} - \Delta t(u_{x,k}\sin\psi_{k|k-1} + u_{y,k}\cos\psi_{k|k-1}) \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} z_{k|k-1} \\ \dot{z}_{k|k-1} \\ b_{\mathrm{bar},k|k-1} \end{bmatrix} = \begin{bmatrix} z_{k-1|k-1} + \dot{z}_{k-1|k-1}\Delta t + \frac{1}{2}(\Delta t)^2(u_{z,k} - G) \\ \dot{z}_{k-1|k-1} + \Delta t(u_{z,k} - G) \\ b_{\mathrm{bar},k-1|k-1} \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} \psi_{k|k-1} \\ \dot{\psi}_{k|k-1} \end{bmatrix} = \begin{bmatrix} \psi_{k-1|k-1} + \Delta t u_{\psi_k} \\ u_{\psi_k} \end{bmatrix} \tag{11}$$

The additional state variable b_{bar} in Equation (10) is used to estimate the high bias of the barometer. However, this bias term is kept constant during the prediction step of the EKF and only updated in the correction step of the barometer, as described in Section 4.2.3. Further, note that the Equations (8) to (10) use the predicted angle $\psi_{k|k-1}$ resulting from Equation (11).

To use the IMU states for the EKF state fusion, the Equations (8) to (11) are rewritten in matrix form, where the right-hand side of the equations decouple the states and inputs. The general form of the IMU state EKF prediction is:

$$\begin{aligned} \hat{\boldsymbol{X}}_{k|k-1} &= \mathbf{F}_k \hat{\boldsymbol{X}}_{k-1|k-1} + \mathbf{W}_k \boldsymbol{U}_k \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{W}_k \mathbf{Q} \mathbf{W}_k^T \end{aligned} \tag{12}$$

To simplify the equations, the state is split into the different directions. Hence, $\hat{\boldsymbol{X}}_{x,k|k-1}$ is the state prediction in x-direction, according to Equation (8). The covariance matrix $\mathbf{P}$ is calculated with matrices that arise from Equation (12) and the diagonal covariance matrix of the IMU sensor $\mathbf{Q}$.

For the x-direction, the matrices $\mathbf{F}_{x,k}$, $\mathbf{W}_{x,k}$, $\mathbf{Q}_x$ and the vector $\boldsymbol{U}_{x,k}$ are:

$$\begin{aligned} \mathbf{F}_{x,k} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} & \mathbf{W}_{x,k} &= \begin{bmatrix} -\frac{1}{2}(\Delta t)^2\cos\psi_{k|k-1} & \frac{1}{2}(\Delta t)^2\sin\psi_{k|k-1} \\ -\Delta t\cos\psi_{k|k-1} & \Delta t\sin\psi_{k|k-1} \end{bmatrix} \\ \mathbf{Q}_x &= \begin{bmatrix} \sigma^2_{\mathrm{imu},x} & 0 \\ 0 & \sigma^2_{\mathrm{imu},y} \end{bmatrix} & \boldsymbol{U}_{x,k} &= \begin{bmatrix} u_{x,k} \\ u_{y,k} \end{bmatrix} \end{aligned} \tag{13}$$

For the y-direction, the matrices $\mathbf{F}_{y,k}$, $\mathbf{W}_{y,k}$, $\mathbf{Q}_y$ and the vector $\boldsymbol{U}_{y,k}$ are:

$$\begin{aligned} \mathbf{F}_{y,k} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} & \mathbf{W}_{y,k} &= \begin{bmatrix} -\frac{1}{2}(\Delta t)^2\sin\psi_{k|k-1} & -\frac{1}{2}(\Delta t)^2\cos\psi_{k|k-1} \\ -\Delta t\sin\psi_{k|k-1} & -\Delta t\cos\psi_{k|k-1} \end{bmatrix} \\ \mathbf{Q}_y &= \begin{bmatrix} \sigma^2_{\mathrm{imu},x} & 0 \\ 0 & \sigma^2_{\mathrm{imu},y} \end{bmatrix} & \boldsymbol{U}_{y,k} &= \begin{bmatrix} u_{x,k} \\ u_{y,k} \end{bmatrix} \end{aligned} \tag{14}$$

For the z-direction, the matrices $\mathbf{F}_{z,k}$, $\mathbf{W}_{z,k}$, $\mathbf{Q}_z$ and the vector $\boldsymbol{U}_{z,k}$ are:

$$\begin{aligned} \mathbf{F}_{z,k} &= \begin{bmatrix} 1 & \Delta t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{W}_{z,k} = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 \sin\psi_{k|k-1} \\ \Delta t \\ 0 \end{bmatrix} \\ \mathbf{Q}_z &= \begin{bmatrix} \sigma^2_{\text{imu},z} \end{bmatrix} \qquad \boldsymbol{U}_{z,k} = \begin{bmatrix} u_{z,k} - G \end{bmatrix} \end{aligned} \tag{15}$$

For the angle ψ, the matrices $\mathbf{F}_{\psi,k}$, $\mathbf{W}_{\psi,k}$, $\mathbf{Q}_\psi$ and the vector $\boldsymbol{U}_{\psi,k}$ are:

$$\mathbf{F}_{\psi,k} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad \mathbf{W}_{\psi,k} = \begin{bmatrix} \Delta t \\ 1 \end{bmatrix} \quad \mathbf{Q}_\psi = \begin{bmatrix} \sigma^2_{\text{imu},\psi} \end{bmatrix} \quad \boldsymbol{U}_{\psi,k} = \begin{bmatrix} u_{\psi,k} \end{bmatrix} \tag{16}$$

Note again, that Equations (13) to (15) use the predicted angle $\psi_{k|k-1}$ resulting from Equation (16).

4.2 Extended Kalman Filter Correction

Due to measurement noise, the position and angle estimates with the IMU measurements drift quickly. Hence, position and angle estimation using an IMU sensor only would not be reliable. To overcome this drift, additional sensors are used. The sensor measurements are fused together with the IMU measurements with an EKF to obtain a more reliable estimate. The additional sensors help to correct the drift of the IMU measurements. The general EKF correction uses the following scheme.

$$\begin{aligned} \boldsymbol{K}_k &= \mathbf{P}_{k|k-1}\boldsymbol{H}_k^T(\boldsymbol{H}_k\mathbf{P}_{k|k-1}\boldsymbol{H}_k^T + \boldsymbol{V}_k\mathbf{R}_k\boldsymbol{V}_k)^{-1} \\ \hat{\boldsymbol{X}}_{k|k} &= \hat{\boldsymbol{X}}_{k|k-1} + \boldsymbol{K}_k\big[\boldsymbol{Y}_k - \boldsymbol{h}(\hat{\boldsymbol{X}}_{k|k-1})\big] \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{H}_k\mathbf{P}_{k|k-1} \end{aligned} \tag{17}$$

In Equation (17), $\boldsymbol{K}_k$ is the Kalman gain, $\boldsymbol{H}_k$ is the Jacobian of the measurement with respect to the robot state, $\boldsymbol{V}_k$ is the Jacobian of the measurement with respect to the raw sensor measurements, $\boldsymbol{Y}_k$ is the sensor measurement, $\boldsymbol{h}$ is the forward sensor model and $\mathbf{R}_k$ is the diagonal covariance matrix of the sensor. The Kalman gain $\boldsymbol{K}_k$ depends on the estimated covariance matrix $\mathbf{P}_{k|k-1}$ which is calculated with the IMU measurements at time step k. Additionally, the forward sensor model $\boldsymbol{h}$ depends on the state estimate $\hat{\boldsymbol{X}}_{k|k-1}$ which is calculated with the IMU measurements.

Therefore, the final state estimate $\hat{\boldsymbol{X}}_{k|k}$ and final covariance matrix $\mathbf{P}_{k|k}$ consist of taking the IMU prediction and correcting it with the sensor measurements to get the state estimation. This gives a precise and reliable final estimate. In case multiple sensor values are received, the corrections are applied one by one and each sensor uses the latest adjusted $\mathbf{P}$ matrix and $\hat{\boldsymbol{X}}$ vector. If no sensor value is received, no correction is applied, and it follows:

$$\begin{aligned} \hat{\boldsymbol{X}}_{k|k} &= \hat{\boldsymbol{X}}_{k|k-1} \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} \end{aligned} \tag{18}$$

Depending on which sensor variances are chosen, the EKF trusts one sensor more or less. This trust value strongly impacts the state estimation of the hector drone. In the following, the different additional sensors and their implementations are discussed.

4.2.1 Global Positioning System

A GPS can estimate the 3D-position of an object by measuring the λ (longitude), ϕ (latitude) and h (height) with geostationary satellites. Since an x, y, z-position estimate in the world frame is needed to operate hector, the satellite measurements need to be transformed into the world coordinate frame. The methodology is to first calculate the geodetic, earth-centered earth-fixed (ECEF) coordinates, then the local north-east-down (NED) coordinates and finally the

coordinates in the world frame. With a being the equatorial radius, b the polar radius and $N(\phi)$ the prime vertical radius of curvature, the ECEF position coordinates are:

$$\begin{aligned} e^2 &= 1 - \frac{b^2}{a^2} \\ N(\phi) &= \frac{a}{\sqrt{1 - e^2 \sin \phi^2}} \\ \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} &= \begin{bmatrix} \big(N(\phi) + h\big) \cos\phi \cos\lambda \\ \big(N(\phi) + h\big) \cos\phi \sin\lambda \\ \frac{b^2}{a^2}\big(N(\phi) + h\big) \sin\phi \end{bmatrix} \end{aligned} \tag{19}$$

After obtaining initial ECEF coordinates x_{e_0}, y_{e_0}, z_{e_0} from a first run, the NED coordinates are found by rotating the ECEF coordinates.

$$\begin{aligned} \mathbf{R}_{e\to n} &= \begin{bmatrix} -\sin\phi\cos\lambda & -\sin\lambda & -\cos\phi\cos\lambda \\ -\sin\phi\sin\lambda & \cos\lambda & -\cos\phi\sin\lambda \\ \cos\phi & 0 & -\sin\phi \end{bmatrix} \\ \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} &= \mathbf{R}_{e\to n}^T \left(\begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} - \begin{bmatrix} x_{e_0} \\ y_{e_0} \\ z_{e_0} \end{bmatrix} \right) \end{aligned} \tag{20}$$

The world frame GPS position estimates are obtained by rotating the NED coordinates and adding these values to the initial world frame position x_0, y_0, z_0 of the hector drone.

$$\begin{aligned} \mathbf{R}_{n\to w} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \\ \begin{bmatrix} x_{\text{gps}} \\ y_{\text{gps}} \\ z_{\text{gps}} \end{bmatrix} &= \mathbf{R}_{n\to w} \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \end{aligned} \tag{21}$$

The GPS position estimates are used to correct the IMU position estimates with the EKF correction equations defined in Equation (17). In order to use these equations, the matrices and scalars $\boldsymbol{Y}_{\text{gps},x,k}$, $\boldsymbol{H}_{\text{gps},x,k}$, $\mathbf{R}_{\text{gps},x,k}$, $\boldsymbol{V}_{\text{gps},x,k}$ and $\boldsymbol{h}(\hat{\boldsymbol{X}}_{x,k|k-1})$ are defined as

$$\begin{aligned} \boldsymbol{Y}_{\text{gps},x,k} &= x_{\text{gps}} \\ \boldsymbol{h}(\hat{\boldsymbol{X}}_{x,k|k-1}) &= x_{k|k-1} \\ \boldsymbol{H}_{\text{gps},x,k} &= \frac{\partial \boldsymbol{h}(\hat{\boldsymbol{X}}_{x,k|k-1})}{\partial \hat{\boldsymbol{X}}_{x,k|k-1}} = \Big[\frac{\partial x_{k|k-1}}{\partial x_{k|k-1}} \ \frac{\partial x_{k|k-1}}{\partial \dot{x}_{k|k-1}}\Big] = \begin{bmatrix} 1 & 0 \end{bmatrix} \\ \boldsymbol{V}_{\text{gps},x,k} &= 1 \\ \mathbf{R}_{\text{gps},x,k} &= \sigma^2_{\text{gps},x} \end{aligned} \tag{22}$$

The forward sensor model $\boldsymbol{h}(\hat{\boldsymbol{X}}_{x,k|k-1}) = x_{k|k-1}$ is equal to the position estimate of the IMU.

For the y- and z-direction, the same values as in Equation (22) can be used by replacing x_{gps} with y_{gps} respectively z_{gps}, and $\sigma^2_{\text{gps},x}$ with $\sigma^2_{\text{gps},y}$ respectively $\sigma^2_{\text{gps},z}$. Additionally, due to the additional bias term for the barometer, $\boldsymbol{H}_{\text{gps},z,k}$ has to be slightly adjusted.

$$\boldsymbol{H}_{\text{gps},z,k} = \frac{\partial \boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1})}{\partial \hat{\boldsymbol{X}}_{z,k|k-1}} = \Big[\frac{\partial z_{k|k-1}}{\partial z_{k|k-1}} \ \frac{\partial z_{k|k-1}}{\partial \dot{z}_{k|k-1}} \ \frac{\partial z_{k|k-1}}{\partial b_{\text{bar},k|k-1}}\Big] = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{23}$$

4.2.2 Magnetometer Sensor

A magnetometer measures the force vector (x_{mgn}, y_{mgn}, z_{mgn}) of magnetic north with a magnetic compass. The x and y component of this force vector are used to estimate the yaw angle ψ.

$$\psi_{\text{mgn}} = \text{atan2}\left(y_{\text{mgn}}, x_{\text{mgn}}\right) \tag{24}$$

The magnetometer yaw angle estimate is used to correct the IMU yaw angle estimate with the EKF correction equations defined in Equation (17). In order to use these equations, the matrices and scalars $\boldsymbol{Y}_{\text{mgn},\psi,k}$, $\boldsymbol{H}_{\text{mgn},\psi,k}$, $\mathbf{R}_{\text{mgn},\psi,k}$, $\boldsymbol{V}_{\text{mgn},\psi,k}$ and $\boldsymbol{h}(\hat{\boldsymbol{X}}_{\psi,k|k-1})$ are defined as

$$\begin{aligned}
\boldsymbol{Y}_{\text{mgn},\psi,k} &= \psi_{\text{mgn}} \\
\boldsymbol{h}(\hat{\boldsymbol{X}}_{\psi,k|k-1}) &= \psi_{k|k-1} \\
\boldsymbol{H}_{\text{mgn},\psi,k} &= \frac{\partial \boldsymbol{h}(\hat{\boldsymbol{X}}_{\psi,k|k-1})}{\partial \hat{\boldsymbol{X}}_{\psi,k|k-1}} = \Big[\frac{\partial \psi_{k|k-1}}{\partial \psi_{k|k-1}} \;\; \frac{\partial \psi_{k|k-1}}{\partial \dot{\psi}_{k|k-1}}\Big] = \begin{bmatrix} 1 & 0 \end{bmatrix} \\
\boldsymbol{V}_{\text{mgn},\psi,k} &= 1 \\
\mathbf{R}_{\text{mgn},\psi,k} &= \sigma^2_{\text{mgn},\psi}
\end{aligned} \tag{25}$$

The forward sensor model $\boldsymbol{h}(\hat{\boldsymbol{X}}_{\psi,k|k-1}) = \psi_{k|k-1}$ is equal to the yaw angle estimate of the IMU.

4.2.3 Barometer Sensor

The barometer determines the height above sea level z_{bar} with pressure measurements. The barometer has a high bias b_{bar}, which must be considered to function correctly.

The barometer altitude estimate is used to correct the IMU z-position estimate with the EKF correction equations defined in Equation (17). In order to use these equations, the matrices and scalars $\boldsymbol{Y}_{\text{bar},z,k}$, $\boldsymbol{H}_{\text{bar},z,k}$, $\mathbf{R}_{\text{bar},z,k}$, $\boldsymbol{V}_{\text{bar},z,k}$ and $\boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1})$ are defined as

$$\begin{aligned}
\boldsymbol{Y}_{\text{bar},z,k} &= z_{\text{bar}} \\
\boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1}) &= z_{k|k-1} + b_{\text{bar},k|k-1} \\
\boldsymbol{H}_{\text{bar},z,k} &= \frac{\partial \boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1})}{\partial \hat{\boldsymbol{X}}_{z,k|k-1}} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \\
\boldsymbol{V}_{\text{bar},z,k} &= 1 \\
\mathbf{R}_{\text{bar},z,k} &= \sigma^2_{\text{bar},z}
\end{aligned} \tag{26}$$

The forward sensor model $\boldsymbol{h}(\hat{X}_{z,k|k-1}) = z_{k|k-1} + b_{\text{bar},k|k-1}$ is equal to the sum of z-position estimate of the IMU and the bias of the barometer.

4.2.4 Sonar Sensor

The sonar sensor uses ultrasound signals to measure the distance to the ground. This can be used to measure the altitude of the hector drone. The maximum range for the sonar is 3 m and it is affected by the heights of ground obstacles. A possible approach to incorporate ground obstacles is outlined in Section 6.

The sonar altitude estimate is used to correct the IMU z-position estimate with the EKF correction equations defined in Equation (17). In order to use these equations, the matrices and scalars $\boldsymbol{Y}_{\text{snr},z,k}$, $\boldsymbol{H}_{\text{snr},z,k}$, $\mathbf{R}_{\text{snr},z,k}$, $\boldsymbol{V}_{\text{snr},z,k}$ and $\boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1})$ are defined as

$$\begin{aligned}
\boldsymbol{Y}_{\text{snr},z,k} &= z_{\text{snr}} \\
\boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1}) &= z_{k|k-1} \\
\boldsymbol{H}_{\text{snr},z,k} &= \frac{\partial \boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1})}{\partial \hat{\boldsymbol{X}}_{z,k|k-1}} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\
\boldsymbol{V}_{\text{snr},z,k} &= 1 \\
\mathbf{R}_{\text{snr},z,k} &= \sigma^2_{\text{snr},z}
\end{aligned} \tag{27}$$

The forward sensor model $\boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1}) = z_{k|k-1}$ is equal to the z-position estimate of the IMU.

4.3 Implementation

Figure 7 shows how the sensor measurements are used to correct the state predictions, which are only based on the IMU measurements. If multiple sensor values are received, the corrections are applied one by one. If no sensor value is received, no correction is applied and the state estimation is only based on the IMU.

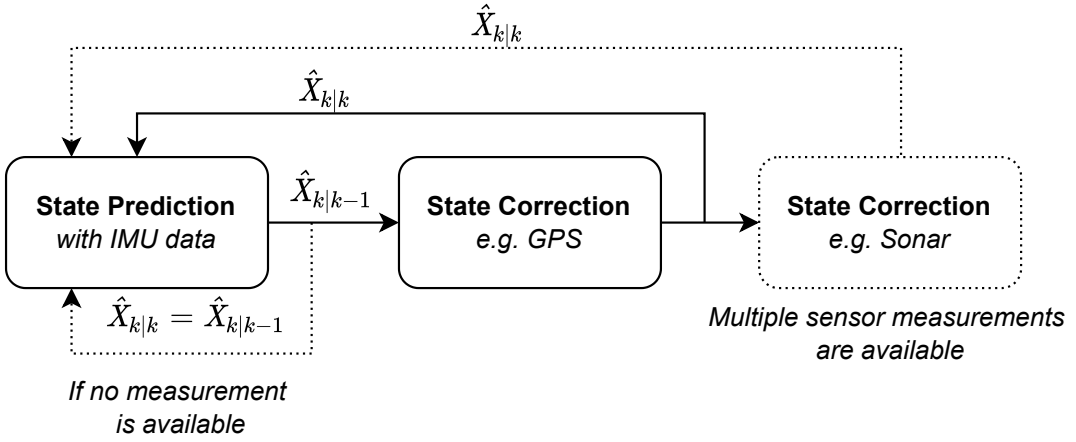


Figure 7: EKF state estimation with multiple sensor measurements.

In practice, the state prediction is implemented in the `cbImu` callback function. The state correction is implemented in the `main` function, iterating over all sensors, checking for new measurements and correcting the state estimation. Consequently, the sensor callback functions `cbGps`, `cbSonar`, `cbBaro` and `cbMagnet` are only used to save the latest sensor measurement $\boldsymbol{Y}_k$. This also includes necessary transformations, such as for the GPS and the magnetometer. Further, the first measurement of the barometer is used to initialize the bias with $b_{\text{bar},0} = z_{\text{bar}} - z_0$.

4.3.1 Magnetometer Correction

As the drone angle is defined from $\psi = [-\pi, \pi]$, the `limit_angle()` was used in the prediction and correction step. Further, it was necessary to adjust the calculation of the magnetometer measurement value $\boldsymbol{Y}_{\text{mgn},\psi,k}$ depending on the current yaw angle $\psi_{k|k-1}$.

$$\boldsymbol{Y}_{\text{mgn},\psi,k} = \begin{cases} \psi_{\text{mgn}}, & \text{if } \psi_{\text{mgn}} - \psi_{k|k-1} \leq \pi \\ \psi_{\text{mgn}} + 2\pi, & \text{otherwise} \end{cases} \tag{28}$$

This prevents a correction into the wrong direction.

4.3.2 Variance Estimation

A reliable state estimate is obtained by finding a good trade-off between the different sensor measurements. The variance σ^2 is a metric to quantify the deviation of the signal, where $\sigma^2 = 0$ indicates a perfect signal with zero noise. The trust in a specific sensor can thus be quantified by its variance: The higher the variance, the less reliable the signal. The variances of each of the sensors are fixed parameters during the simulation and can be estimated from the sensor data y using Equation (29).

$$\sigma^2 = \frac{\sum \left(y - \frac{\Sigma y}{N}\right)^2}{N} = \frac{\sum \left(y - \mu\right)^2}{N} \tag{29}$$

The measurement data from each of the sensors is gathered during a steady state flight, where the state variables have constant mean. The variances have to be computed from the corresponding state. Since the IMU measures accelerations, for this particular sensor, we first need to compute the estimate following Equations (13) and (15). In order to fully decouple the sensor, the current estimate of the IMU is computed exclusively from the previous IMU estimate. Similarly, the GPU measurements λ, ϕ and h are used to compute the GPS estimate following the equations in Section 4.2.1. For the rest of the sensors (SNR, MGN, BAR), the state is directly obtained from the measurement data.

After running the simulation with the computed values, it was observed that the IMU measurements were very imprecise due to the high drift. Further, the angle predictions strongly degraded during acceleration and deceleration of the drone. Hence, an additional scaling factor $\alpha = 10$ for $\sigma_x^2, \sigma_y^2, \sigma_z^2$ and $\alpha = 1000$ for σ_ψ^2 was applied to compensate these effects. The resulting values are shown in Table 2.

Table 2: The variances for the different sensors.

Sensor	σ_x^2	σ_y^2	σ_z^2	σ_ψ^2
IMU	0.26052	0.11307	0.06024	0.007
GPS	0.002471	0.012065	0.004479	-
MGN	-	-	-	0.000182
BAR	-	-	0.034431	-
SNR	-	-	0.000027	-

5 Simulation Results

In this section, observations and results of the final implementation are discussed. For the final simulation, the splines, EKF and PID controller are enabled simultaneously. The presented data is consistent with the provided video and code, and the parameters chosen correspond to the ones in the submitted `hector.yaml` file. Furthermore, a comparative study was conducted to gain a better understanding of the individual effect of each of the correction terms in the EKF.

Figure 8 shows the measurements of the states for the final simulation. It is to be noted that the IMU graph is not purely the integrated IMU data because the IMU prediction is computed based on the previous state estimate which is influenced by the sensor corrections. The x- and y-position predictions get corrected by the GPS sensor while the z-position predictions are corrected by combining the GPS, SNR and BAR sensors. Finally, the magnetometer corrects the yaw angle ψ prediction.

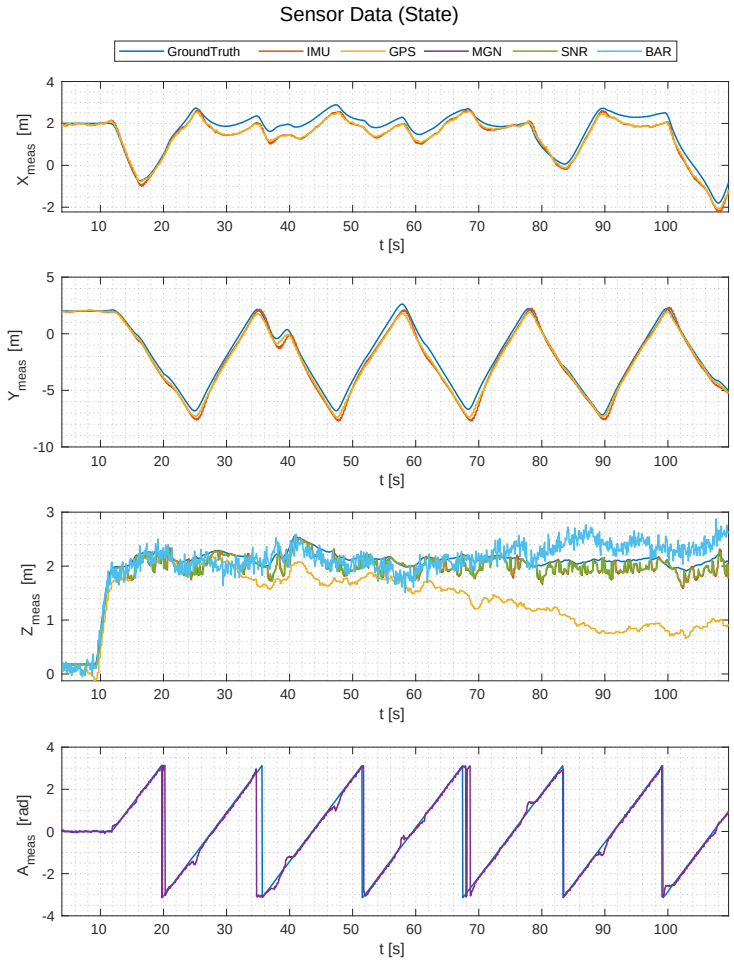


Figure 8: Measured data obtained from sensors, before computing correction terms.
(The yaw angle ψ is labelled with the letter A)

5.1 Bias Estimation for Barometer

As mentioned in Section 4.2, the barometer has a strong bias, which can be predicted by augmenting the state estimation to include the bias term $b_{\text{bar},k|k}$. Figure 9 shows two different implementations of the barometer correction by comparing $b_{\text{bar},k|k}$ to the ground truth bias $b_{\text{bar,truth}} = z_{\text{bar}} - z_{\text{truth}}$. For this comparison, a high value is used for the initial bias $b_{\text{bar},0}$ and

then the evolution of the estimate over time is analyzed. The dynamics of the bias are expected to converge to a steady state value. This intuition is used to evaluate the correctness of our implementation. According to the project documentation $\boldsymbol{H}_{\text{bar},z,k}$ is given by

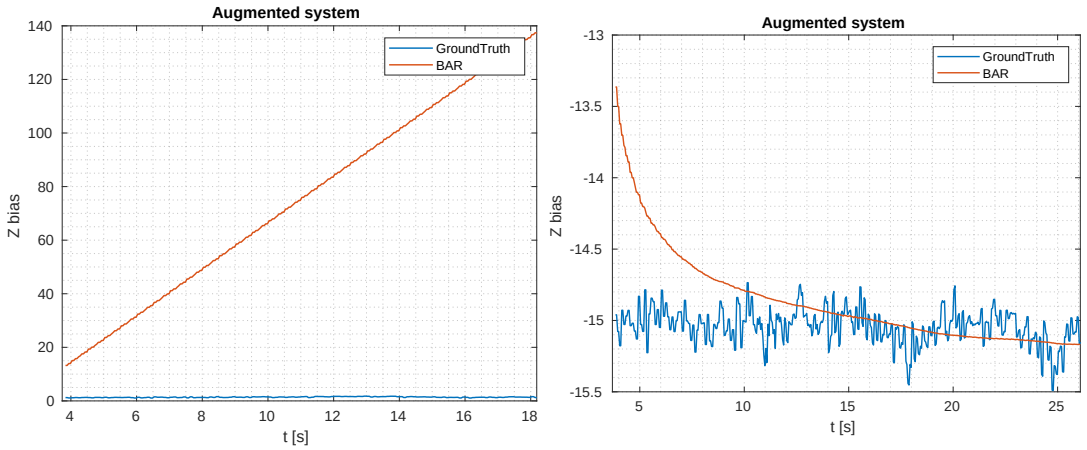
$$\boldsymbol{H}_{\text{bar},z,k} = \frac{\partial \boldsymbol{Y}_{\text{bar},z,k}}{\partial \hat{\boldsymbol{X}}_{z,k|k-1}} \tag{30}$$

Hence, originally $\boldsymbol{H}_{\text{bar},z,k} = [1\ 0\ -1]$ was chosen. However, this led to the system diverging in $b_{\text{bar},k|k-1}$, as shown in Figure 9a.

It is proposed to define $\boldsymbol{H}_{\text{bar},z,k}$ as

$$\boldsymbol{H}_{\text{bar},z,k} = \frac{\partial \boldsymbol{h}(\hat{\boldsymbol{X}}_{z,k|k-1})}{\partial \hat{\boldsymbol{X}}_{z,k|k-1}} \tag{31}$$

This yields the sensor equation stated in Equation (26) in Section 4.2.3 and also complies with the lecture and secondary material. Consequently, this gives $\boldsymbol{H}_{\text{bar},z,k} = [1\ 0\ 1]$ and leads to a stable behaviour of the bias term. Thus, the definition in eq. (31) was chosen in the final implementation. Figure 9b shows the results with this implementation.



(a) $b_{\text{bar},k|k}$ when using $\boldsymbol{H}_{\text{bar},z,k} = [1\ 0\ -1]$

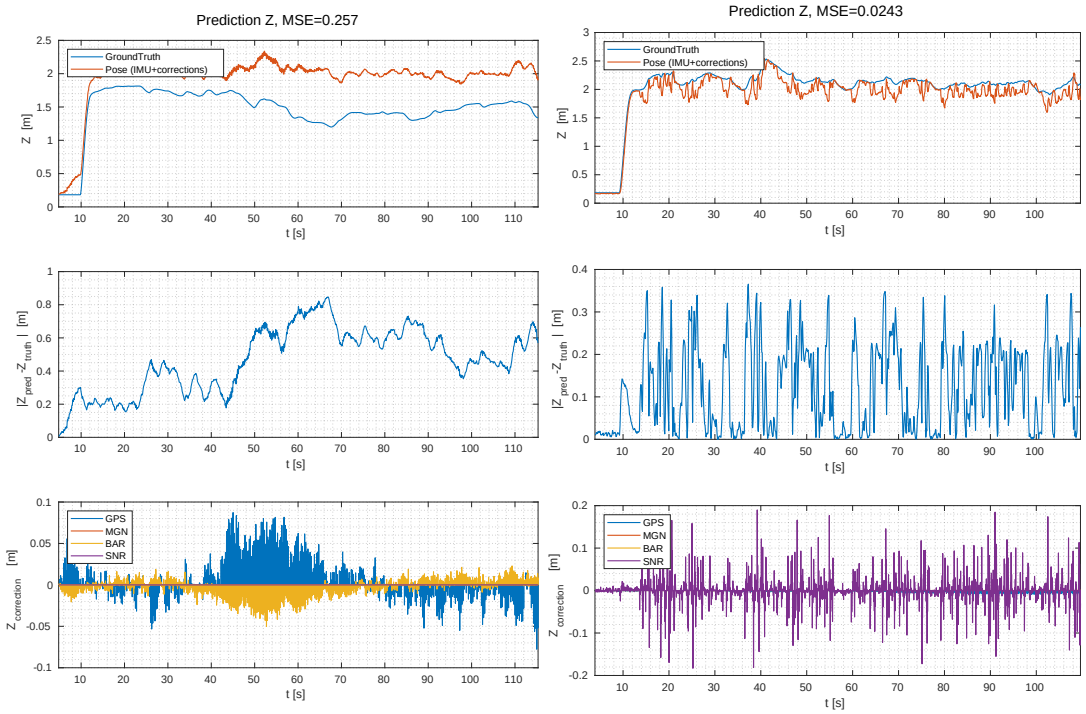
(b) $b_{\text{bar},k|k}$ when using $\boldsymbol{H}_{\text{bar},z,k} = [1\ 0\ 1]$

Figure 9: Comparison of different implementations for bias estimation of barometer

5.2 Barometer vs. Sonar Correction

Both the barometer and sonar sensors contribute to the correction of the height $\hat{\boldsymbol{X}}_{z,k|k-1}$. From the variance computation, it's clear that the sonar sensor is more precise and thus should be the main contributor to the height estimation. To understand the interplay between both sensors, a simulation where $\hat{\boldsymbol{X}}_{z,k|k}$ is estimated solely from the GPS and barometer was first carried out (see Figure 10a). The results are compared to a second simulation, which includes the sonar correction (see Figure 10b).

Figure 10 shows the estimated height $z_{k|k}$ after applying the corrections to the IMU, along with the ground truth. In the middle plot, the absolute error is computed. The bottom plot shows the correction terms which are computed as explained in Section 4.2. While the barometer provides a better estimate than the GPS, a significant improvement is obtained when introducing the sonar correction. The mean square error is used as the comparison metric.



(a) Height estimation $z_{k|k}$ obtained from GPS and barometer.

(b) Height estimation $z_{k|k}$ obtained from GPS, barometer and sonar.

Figure 10: Comparison between height prediction using barometer and sonar sensors. First row: height prediction and ground truth over time. Second row: error of predicted height over time. Third row: height corrections from sensors over time.

5.3 Bias Estimation for IMU

It was noted that the IMU acceleration measurements contain a high drift. The effect can be damped by estimating a bias for the acceleration terms. Since the drift depends on the specific simulation instance, the estimation is done at the start of the simulation. A delay of 10 s is introduced to induce a steady state before the drone starts to fly. This allows to gather enough initial data points during steady state, from which the bias is computed by averaging over the collected points. Once the drone starts to fly, the estimated bias is kept constant for the remainder of the simulation.

Different experiments were conducted, and it was noted that the bias for u_x, u_y, u_z are always around $0.1\,\mathrm{m\,s^{-2}}$. Figure 11 shows the measured IMU accelerations for the x-, y- and z-direction and the measured angular velocity as well as the estimated biases for that particular flight. It can be seen that the drone is in steady state at the beginning of the simulation ($t < 10s$).

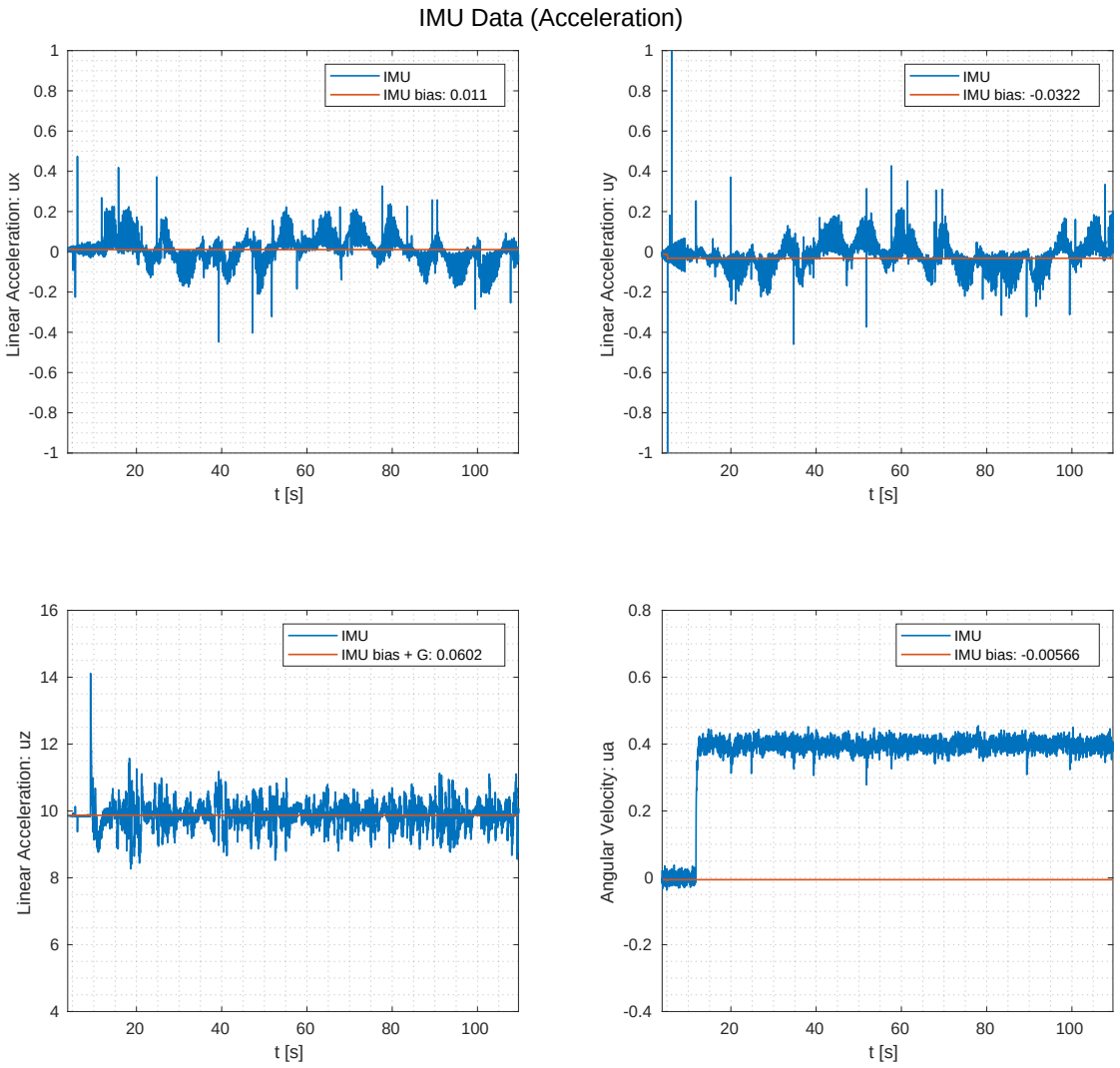


Figure 11: Estimation of IMU bias obtained by averaging over the steady state data points.

5.4 Further Discussion

Following the same plot description as introduced previously, Figure 12 shows the state estimates for $x_{k|k}, y_{k|k}, z_{k|k}, \psi_{k|k}$ obtained by combining all the available sensors during the final simulation. In all the four cases, the state estimate obtained after applying the correction terms to the IMU is consistent with the ground truth. The absolute errors lie within an acceptable range. Furthermore, it can be observed that the sonar sensor is the main contributor to the correction z-direction, while the correction of the yaw angle ψ is mainly driven by the magnetometer.

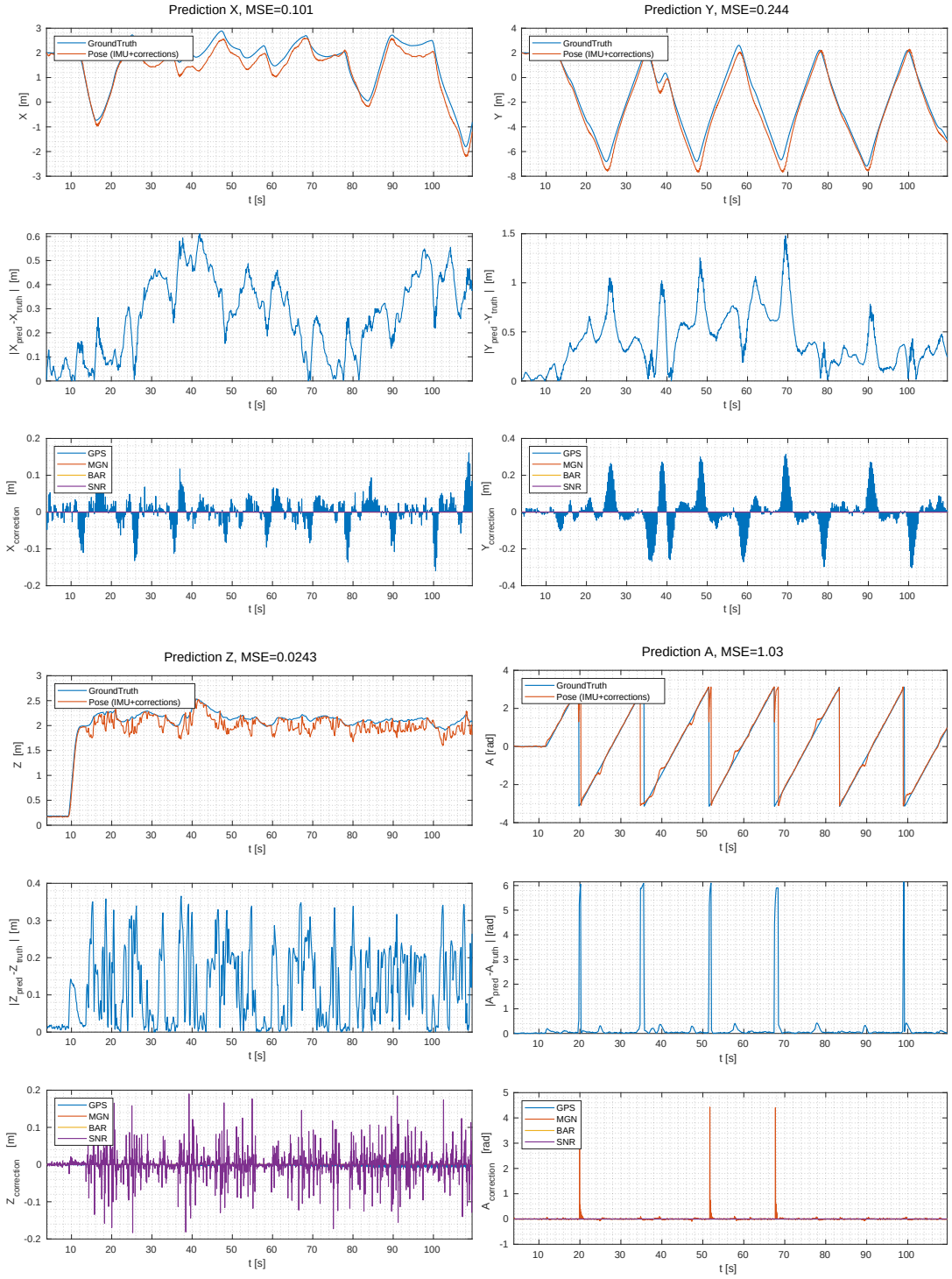


Figure 12: Pose estimation $x_{k|k}, y_{k|k}, z_{k|k}, \psi_{k|k}$ obtained from combining GPS, magnetometer, sonar and barometer corrections. (The yaw angle ψ is labelled with the letter A)

6 Conclusion

The overall performance of the state estimation using the EKF filter is very good and accurate. Consequently, the hector drone is able to continuously fly from the starting point to the moving turtle ground robot, from the turtle to the turtle's final goal and from the final goal back to hector's starting position.

6.1 Key Learnings

During this project, some important lessons were learned. Firstly, implementation errors can quickly lead to an unstable estimation. The IMU prediction was extremely unreliable due to the high drift, which made it very difficult to get a good estimate in the x- and y-direction. This problem was solved by estimating the drift from the measured data at the start of the simulation. Originally, correction terms were implemented independently of each other, so they all relied solely on the IMU. This did not work, because the individual corrections were overlapping one another, which led to divergence of the state estimate. Another problem which appeared was that the barometer was not augmented properly, and the covariance matrix for the bias estimation needed to be initialized with a non-zero value. Further, initially the H matrix was computed wrongly, making the estimate of the bias term diverge. Finally, some problems were encountered with finding the right implementation for the angle calculations. This was solved by closely examining the angles and adding the necessary constraints.

6.2 Future Work

As the sonar sensor measures the distance to the ground, this measurement can be influenced by obstacles. In order to compensate for obstacles, two separate or combined approaches could be used. First, a bias term similar to the one used for the barometer could be added to the state estimation.

$$\hat{\boldsymbol{X}}_z = \begin{bmatrix} z, & \dot{z}, & b_{\text{bar}}, & b_{\text{snr}} \end{bmatrix}^\intercal \tag{32}$$

However, as the real bias of the sensor under normal conditions is low frequent and has a low variance, the convergence to a new value in case of a sudden obstacle would be quite slow. Another approach is to use an additional algorithm to detect sudden steps in the distance measurement. If the rate of change in height is above a certain threshold, the change can be integrated to estimate the height of the step caused by the obstacles. This step detection algorithm can be added to the prediction step of the EKF to fuse both approaches.

Further, the bias of the IMU in all three directions could also be included in the EKF, similar to the method mentioned above.