1. Supplemental Information

Appendix A. Sensor Synchronization and Kalman Filter Models

Linear Interpolation for Sensor Data Synchronization

Since one IMU is also active and exhibits a higher sampling rate than the always-on RGB-D camera, it is selected as the reference sensor. Linear interpolation is utilized to estimate the readings of other sensors at the IMU's sampling points. Consider two sensors: Sensor Z (e.g., RGB-D camera) with a lower sampling rate, and the IMU with a higher sampling rate. Suppose Sensor Z has readings at times t_1 and t_2 (where $t_1 < t_2$) as $Z(t_1)$ and $Z(t_2)$, respectively. The objective is to estimate Sensor Z's reading at time t (where $t_1 < t < t_2$) to synchronize it with the IMU reading at time t.

The linear interpolation formula for Sensor Z at time t is expressed as:

$$Z(t) = Z(t_1) + \frac{(t - t_1)}{(t_2 - t_1)} \times [Z(t_2) - Z(t_1)] \tag{1}$$

where:

- $Z(t)$ is the estimated reading of Sensor A at time t.
- $Z(t_1)$ and $Z(t_2)$ are the actual readings of Sensor Z at times t_1 and t_2, respectively.
- t is the time at which Sensor Z's reading is estimated, aligned with the IMU's sampling point.

This interpolation method allows for the estimation of Sensor Z's readings at intermediate times between its actual sampling points, enabling synchronization with the higher frequency IMU data. Note that the EKF's sampling rate is also set to 100 Hz. The time complexity of this operation for each sensor is O(1).

(Baseline) Kalman Filter Elements:

$\mathbf{x}_k$ State Vector: Represents dynamic variables like position and velocity.
$\mathbf{F}_k$ State Transition Matrix: Defines system dynamics.
$\mathbf{u}_k$ Control Input Vector: Accounts for external control inputs.
$\mathbf{G}_k$ Control Matrix: Links control inputs to the state model.
$\mathbf{z}_k$ Observation Vector: Sensor measurements.
$\mathbf{C}_k$ Observation Matrix: Relates state vector to observations.
$\mathbf{Q}_k$ Process Noise Covariance Matrix: Uncertainty in the process model.
$\mathbf{R}_k$ Measurement Noise Covariance Matrix: Uncertainty in sensor measurements.
$\mathbf{x}_{0|0}$ Initial State: Starting state vector.
$\mathbf{P}_{0|0}$ Initial Covariance Matrix: Initial uncertainty.
$\mathbf{K}_k$ Kalman Gain: Adjusts the estimate during the measurement update.
$\mathbf{y}_k$ Innovation: is the innovation (measurement residual).
$\mathbf{S}_k$ Innovation Covariance: is the innovation covariance.
Prediction Equations For predicting next state ($\mathbf{x}_{k|k-1}$) and covariance ($\mathbf{P}_{k|k-1}$).
Measurement Update Equations For updating state ($\mathbf{x}_{k|k}$) and covariance ($\mathbf{P}_{k|k}$) with measurements.

Drag Force F_{drag} with density ρ, frontal area A, and drag coefficient C_d: This equation models the drag force acting on the vehicle in the ISS environment

$$F_{\text{drag}} = -0.5 \cdot \rho \cdot A \cdot C_d \cdot v^2 \tag{2}$$

State vector x_k: The state vector includes position and velocity components in 3D space to track the vehicle's movement. Ultrasonic measurements need to be handled differently, and the details for this representation are at the end of this section.

$$\mathbf{x}_k^T = \begin{pmatrix} x_k & y_k & z_k & v_{x_k} & v_{y_k} & v_{z_k} \end{pmatrix} \tag{3}$$

where

- x, y, z are the Cartesian coordinates.
- v_x, v_y, v_z are the velocity components.

Initialization of the Kalman Filter: For the effective implementation of the Kalman Filter, the initialization of the state vector $\mathbf{x}_{0|0}$ and the covariance matrix $\mathbf{P}_{0|0}$ is critical. Given the context of our application, where the initial position is known with 100% accuracy, the initializations are as follows:

Initial State Vector $\mathbf{x_{0|0}}$: The initial state vector is set to represent the system's known starting position and velocity. Since the initial position is controlled and explicitly defined, the initial state vector and covariance are initialized to zeros.

$$\mathbf{x}_{0|0} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T \tag{4}$$

$$\mathbf{P}_{0|0} = \text{zero matrix} \tag{5}$$

This initialization approach ensures a precise starting point for the KF, contributing to the accuracy and stability of the state estimation process in the initial phases of the system's operation. In our system, the initial state estimation depends on the starting true position to be known.

Observation Vector z_k: The observation vector represents measurements obtained from active sensors.

$$\mathbf{z}_k^T = \begin{pmatrix} z_1 & z_2 & \dots & z_n \end{pmatrix} \tag{6}$$

The observation vector represents the sensor measurements and is related to the state vector through the observation matrix $\mathbf{C}_k$ and the observation noise $\mathbf{v}_k$.

$$\mathbf{z}_k = \mathbf{C}_k \mathbf{x}_k + \mathbf{v}_k \tag{7}$$

where $\mathbf{v}_k$ is the observation noise, assumed to be zero-mean Gaussian white noise with covariance matrix $\mathbf{R}_k$, $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$.

Observation Matrix C_k: The observation matrix $\mathbf{C}_k$ relates the state vector to the observation space. Each row of the matrix corresponds to a sensor's measurement model.

$$\mathbf{C}_k = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nm} \end{pmatrix} \tag{8}$$

Where n is the number of sensors and m is the size of the state vector.

State-space representation with drag, microgravity, and sensor feedback: In this model, the state-space representation includes the effects of drag, microgravity, and sensor feedback. The observation model is implicitly included in the state transition, reflecting the direct impact of sensor measurements on the state evolution.

$$\mathbf{x}_{k+1} = \mathbf{F}_k(\mathbf{x}_k, \mathbf{z}_k)\mathbf{x}_k + \mathbf{G}_k\mathbf{u}_k \tag{9}$$

State transition matrix F_k: The state transition matrix models the vehicle's dynamics, incorporating the effects of drag.

$$\mathbf{F}_k = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ -\frac{C_d A \rho \Delta t}{2m} & 0 & 0 & 1 & 0 & 0 \\ 0 & -\frac{C_d A \rho \Delta t}{2m} & 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{C_d A \rho \Delta t}{2m} & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

Control input vector u_k in transpose form: This vector represents the external control inputs to the system as affected by thrust Θ and microgravity μ_g environment.

$$\mathbf{u}_k^T = \begin{pmatrix} \Theta_x + \mu_g & \Theta_y + \mu_g & \Theta_z + \mu_g \end{pmatrix} \tag{11}$$

Control matrix G_k: The control matrix links the control input to the state-space model.

$$\mathbf{G}_k = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{pmatrix} \tag{12}$$

Prediction Step: These equations represent the prediction phase of the Kalman Filter, forecasting the next state and estimating error covariance.

$$\mathbf{x}_{k|k-1} = \mathbf{F}_k \mathbf{x}_{k-1|k-1} + \mathbf{G}_k \mathbf{u}_k \tag{13}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{14}$$

Update Step: The update phase of the Kalman Filter refines the predictions based on new measurements from various sensors, incorporating the observation model.

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k \mathbf{x}_{k|k-1} \tag{15}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{16}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{17}$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k \tag{18}$$

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{19}$$

where $\mathbf{y}_k$ is the innovation or measurement residual, $\mathbf{S}_k$ is the innovation covariance, and $\mathbf{K}_k$ is the Kalman Gain.

Ultrasonic Sensor State Space Model

Ultrasonic sensors measure the time taken for an ultrasonic pulse to reflect back from an object. The state space model for an ultrasonic sensor can be described as follows:

Observation Vector for Ultrasonic Sensor $z_{k,\mathrm{US}}$:

$$z_{k,\mathrm{US}} = \frac{2d_k}{c} \tag{20}$$

where d_k is the distance to the object, and c is the speed of sound. The Observation Matrix places $\frac{1}{c}$ along the main diagonal to project the state vector into the measurement space of the ultrasonic sensor.

Transition from KF to EKF

KF to EKF State Vector Inheritance: The state vector of the EKF is initialized with the last estimated state vector from the KF to ensure continuity. The transition formula is as follows:

$$\{x, y, z, v_x, v_y, v_z\}_k^{\text{EKF}} = \{x, y, z, v_x, v_y, v_z\}_k^{\text{KF}} \tag{21}$$

The transition from the KF to the EKF involves adapting the state estimation process to account for non-linear system dynamics and observation models. This is achieved by:

- Inheriting the state vector from the KF as the initial state for the EKF.
- Defining the non-linear system dynamics function $f(\mathbf{x}_k, \mathbf{u}_k)$ and the observation model $h(\mathbf{x}_k)$.
- Computing the Jacobians of these functions, $\mathbf{F}_k$ and $\mathbf{H}_k$, to linearize them around the current state estimate.
- Implementing the EKF prediction and update steps using these non-linear functions and their Jacobians.

Extended Kalman Filter (EKF)

Overview: The Extended Kalman Filter (EKF) is an extension of the standard KF, designed to handle non-linear system dynamics and observation models through linearization techniques. It is particularly useful in scenarios where the process or measurement models do not follow linear behavior or a high amount of uncertainty exists.

EKF Variables with Definitions:

$f()$ is the non-linear function that describes the system dynamics.
$h()$ is the non-linear function that describes the observation model.
$\mathbf{F}_k$ is the Jacobian of $f()$, evaluated at the current state and control input.
$\mathbf{H}_k$ is the Jacobian of $h()$, evaluated at the current state.
$\mathbf{Q}_k$ is the process noise covariance matrix.
$\mathbf{R}_k$ is the measurement noise covariance matrix.
$\mathbf{K}_k$ is the Kalman gain.
$\mathbf{y}_k$ is the innovation (measurement residual).
$\mathbf{S}_k$ is the innovation covariance.
$\mathbf{P}_k$ is the state covariance matrix.

State Vector (EKF) with Definitions: The state vector in EKF represents the estimated state of the system, including both position and velocity components in 3D space.

$$\mathbf{x}_k^{\text{EKF},T} = \begin{pmatrix} x & y & z & v_x & v_y & v_z \end{pmatrix} \tag{22}$$

The control input vector, observation vector, and observation matrix have the same structure and definitions from the KF and are omitted here (please see above for reference).

Prediction and Update Equations for EKF: The EKF prediction and update equations are designed to estimate the state of the system at the next time step, taking into account the non-linear dynamics.

Prediction (EKF): In the prediction phase, the EKF estimates the state of the system at the next time step. This involves using a non-linear function $f()$ to predict the next state based on the current state and control inputs.

$$\hat{\mathbf{x}}_{k+1|k}^{\text{EKF}} = f(\hat{\mathbf{x}}_{k|k}^{\text{EKF}}, \mathbf{u}_k) \tag{23}$$

The uncertainty associated with the state prediction is also updated. This is done by applying the Jacobian of the system dynamics function ($\mathbf{F}_k$) to the current state covariance, along with adding process noise ($\mathbf{Q}_k$).

$$P_{k+1|k}^{\text{EKF}} = \mathbf{F}_k P_{k|k}^{\text{EKF}} \mathbf{F}_k^T + \mathbf{Q}_k \tag{24}$$

Update (EKF): In the update phase, the EKF refines its prediction based on new sensor measurements. The difference between the actual measurements and what the model predicted is calculated, known as the innovation.

$$\mathbf{y}_{k+1} = \mathbf{z}_{k+1} - h(\hat{\mathbf{x}}_{k+1|k}^{\text{EKF}}) \tag{25}$$

The innovation covariance ($\mathbf{S}_{k+1}$) is then calculated, which measures the confidence in the predicted state.

$$\mathbf{S}_{k+1} = \mathbf{H}_{k+1} P_{k+1|k}^{\text{EKF}} \mathbf{H}_{k+1}^T + \mathbf{R}_{k+1} \tag{26}$$

The Kalman gain ($\mathbf{K}_{k+1}$), which determines how much to adjust the prediction based on the new measurement, is computed next.

$$\mathbf{K}_{k+1} = P_{k+1|k}^{\text{EKF}} \mathbf{H}_{k+1}^T \mathbf{S}_{k+1}^{-1} \tag{27}$$

The state estimate is updated using this Kalman gain, applying it to the innovation.

$$\hat{\mathbf{x}}_{k+1|k+1}^{\text{EKF}} = \hat{\mathbf{x}}_{k+1|k}^{\text{EKF}} + \mathbf{K}_{k+1} \mathbf{y}_{k+1} \tag{28}$$

Finally, the state covariance matrix is updated, which reflects the new level of confidence in the state estimate after considering the latest measurement.

$$P_{k+1|k+1}^{\text{EKF}} = (I - \mathbf{K}_{k+1} \mathbf{H}_{k+1}) P_{k+1|k}^{\text{EKF}} \tag{29}$$

Unscented Kalman Filter (UKF)

Overview: The Unscented Kalman Filter (UKF) is an advanced form of the Kalman Filter designed to handle non-linear system dynamics more effectively. Unlike the EKF,

which linearizes non-linear functions through Jacobians, the UKF employs a deterministic sampling technique called Sigma Points. These points capture the mean and covariance of the state estimate and are propagated through the non-linear system, offering a more accurate representation of the state distribution. The UKF is particularly advantageous in systems where the non-linearity is severe or complex, and linearization methods like those used in the EKF might lead to significant estimation errors.

EKF to UKF Transition:

The Unscented Kalman Filter (UKF) inherits the initial state $\mathbf{x}_{\text{initial}}$ and covariance matrix $\mathbf{P}_{\text{initial}}$ from the KF or EKF (described using EKF variables below).

$$\mathbf{x}_{\text{UKF, initial}} = \mathbf{x}_{\text{EKF, final}}, \quad \mathbf{P}_{\text{UKF, initial}} = \mathbf{P}_{\text{EKF, final}} \tag{30}$$

Here, $\mathbf{x}_{\text{EKF, final}}$ and $\mathbf{P}_{\text{EKF, final}}$ are the final state and covariance matrix from the EKF, which serve as the initial conditions for the UKF. These initial conditions are then used to generate the Sigma Points $\mathcal{X}_{\text{initial}}$ and proceed with the UKF's predict-update cycle.

The Sigma Points are especially beneficial for capturing higher-order system non-linearity without requiring explicit differentiation of the system equations, which is one of the main distinctions and advantages of UKF over EKF.

UKF Variables and Functions:

- $\mathcal{X}_{k|k-1}$: Set of Sigma Points at time k based on previous state $k-1$.
- $\mathcal{X}'_{k|k-1}$: Predicted Sigma Points at time k after applying the process model.
- $\mathcal{Y}_k$: Predicted measurements corresponding to $\mathcal{X}'_{k|k-1}$.
- $\hat{\mathbf{x}}_{k|k-1}$: Predicted state vector at time k based on Sigma Points.
- $\hat{\mathbf{y}}_k$: Predicted measurement at time k based on Sigma Points.
- $\mathbf{P}_{k|k-1}$: Predicted error covariance matrix at time k.
- $\mathbf{P}_{yy}$: Covariance of predicted measurements.
- $\mathbf{P}_{xy}$: Cross-covariance between state and measurements.
- $\mathbf{K}_k$: Kalman Gain at time k.
- $\mathbf{W}_m$: Weights for computing the mean.
- $\mathbf{W}_c$: Weights for computing the covariance.
- $\mathbf{u}_k$: Control input vector at time k.
- $\mathbf{z}_k$: Actual measurement at time k.
- $\mathbf{Q}_k$: Process noise covariance matrix at time k.
- $\mathbf{R}_k$: Measurement noise covariance matrix at time k.
- λ: Scaling parameter for Sigma Point generation.
- $\mathbf{f}(\cdot)$: Non-linear state transition function.
- $\mathbf{h}(\cdot)$: Non-linear measurement function.
- GenerateSigmaPoints$(\cdot)$: Function to generate Sigma Points.
- WeightedMean$(\cdot)$: Function to compute weighted mean.
- WeightedCovariance$(\cdot)$: Function to compute weighted covariance.

UKF Function Descriptions:

- **GenerateSigmaPoints**$(\mathbf{x}, \mathbf{P}, \lambda)$: This function generates the Sigma Points $\mathcal{X}$ around the state $\mathbf{x}$ with covariance $\mathbf{P}$ and scaling parameter λ.

$$\mathcal{X}_0 = \mathbf{x}, \quad \mathcal{X}_i = \mathbf{x} + (\sqrt{(n+\lambda)\mathbf{P}})_i, \quad \mathcal{X}_{i+n} = \mathbf{x} - (\sqrt{(n+\lambda)\mathbf{P}})_i \tag{31}$$

where $i = 1, 2, ..., n$ and n is the dimension of the state vector. The matrix square root is typically computed using the Cholesky decomposition.

- **WeightedMean**$(\mathcal{X}, \mathbf{W}_m)$: Calculates the weighted mean of the Sigma Points $\mathcal{X}$ using weights $\mathbf{W}_m$.

$$\hat{\mathbf{x}} = \sum_{i=0}^{2n} \mathbf{W}_{m,i} \mathcal{X}_i \tag{32}$$

- **WeightedCovariance**$(\mathcal{X}, \hat{\mathbf{x}}, \mathbf{W}_c)$: Computes the weighted covariance of the Sigma Points $\mathcal{X}$ relative to the weighted mean $\hat{\mathbf{x}}$ with weights $\mathbf{W}_c$.

$$\mathbf{P} = \sum_{i=0}^{2n} \mathbf{W}_{c,i} (\mathcal{X}_i - \hat{\mathbf{x}})(\mathcal{X}_i - \hat{\mathbf{x}})^T \tag{33}$$

These UKF functions are vital for capturing the non-linear dynamics of the system and sensor measurements. The generation and manipulation of Sigma Points through these functions enable the UKF to approximate the state distribution more accurately than the EKF, especially in systems with significant non-linearities.

UKF Sigma Point Generation Sigma Points are generated to capture the mean and covariance of the state estimate. This step is crucial for the UKF's ability to approximate non-linear transformations of the state.

$$\mathcal{X}_{k|k-1} = \text{GenerateSigmaPoints}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}, \lambda) \tag{34}$$

UKF Prediction Step: Each Sigma Point is propagated through the non-linear state transition function to predict the next state.

$$\mathcal{X}'_{k|k-1} = \mathbf{f}(\mathcal{X}_{k|k-1}, \mathbf{u}_k) \tag{35}$$

The predicted state is then obtained by calculating the weighted mean of the transformed Sigma Points.

$$\hat{\mathbf{x}}_{k|k-1} = \text{WeightedMean}(\mathcal{X}'_{k|k-1}, \mathbf{W}_m) \tag{36}$$

The predicted error covariance is calculated by the weighted covariance of the Sigma Points, adding the process noise.

$$\mathbf{P}_{k|k-1} = \text{WeightedCovariance}(\mathcal{X}'_{k|k-1}, \hat{\mathbf{x}}_{k|k-1}, \mathbf{W}_c) + \mathbf{Q}_k \tag{37}$$

UKF Update Step: Sigma Points are transformed through the measurement function to predict the observation.

$$\mathcal{Y}_k = \mathbf{h}(\mathcal{X}'_{k|k-1}) \tag{38}$$

The predicted measurement is the weighted mean of these transformed points.

$$\hat{\mathbf{y}}_k = \text{WeightedMean}(\mathcal{Y}_k, \mathbf{W}_m) \tag{39}$$

The covariance of the predicted measurement is computed, adding the measurement noise.

$$\mathbf{P}_{yy} = \text{WeightedCovariance}(\mathcal{Y}_k, \hat{\mathbf{y}}_k, \mathbf{W}_c) + \mathbf{R}_k \tag{40}$$

The cross-covariance between the state and the measurements is also computed.

$$\mathbf{P}_{xy} = \text{WeightedCovariance}(\mathcal{X}'_{k|k-1}, \hat{\mathbf{x}}_{k|k-1}, \mathcal{Y}_k, \hat{\mathbf{y}}_k, \mathbf{W}_c) \tag{41}$$

The Kalman Gain, which determines how much the predictions should be adjusted based on the new measurements, is then calculated.

$$\mathbf{K}_k = \mathbf{P}_{xy}\mathbf{P}_{yy}^{-1} \tag{42}$$

The state estimate is updated using the Kalman Gain and the difference between the actual and predicted measurements.

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{y}}_k) \tag{43}$$

Finally, the error covariance matrix is updated.

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{P}_{yy}\mathbf{K}_k^T \tag{44}$$

Transition from UKF to EKF/KF and EKF to KF

- State and Covariance Transfer: The final state ($\mathbf{x}_{\text{UKF, final}}$) and covariance matrix ($\mathbf{P}_{\text{UKF, final}}$) from UKF are used as initial conditions for transitioning to EKF or KF. Similarly, the EKF's final state and covariance can be used as initial conditions for KF.
- Decision Criteria: Transition based on system dynamics becoming more linear, computational efficiency considerations, or improved sensor accuracy.
- Resetting Filters: Resuming EKF or KF with transferred state and covariance, adjusting necessary parameters.

Appendix B. Decision Tree and Graph

Algorithm 1 Decision Trees for Sensor and Filter Activation

procedure SensorActivationDecisionTree(scenario, awareness_scores, current_power, current_cpu, distance_from_X)
    Initialize all component activations to 0
    $camera\_active, imu\_active = 1, 1$ ▷ Always ON
    if `scenario == 'non_critical'` **then**
        if `dist['goal'] > dist_thresh['goal']` **then**
            $kf\_active = 1$ if $awareness\_scores['global'] < awareness\_threshold$
        end if
    else if `scenario == 'moderately_critical'` **then**
        if `dist['obstacle'] < dist_thresh['obstacle']` **then**
            Activate EKF, Ultrasonic, IMU-Mesh based on quality data and current thresholds
        end if
    else if `scenario == 'highly_critical'` **then**
        if `dist['environment'] < dist_thresh['environment']` **then**
            Activate IR, UKF, ToF, IMU-Mesh, Ultrasonic
            Prioritize based on CPU utilization, power usage, and heat generation
        end if
    end if
    Update current_power and current_cpu dictionaries based on activated components
    Update awareness_scores based on activated components and their impact
    return $camera\_active, imu\_active, ir\_active, tof\_active,$
        $kf\_active, ekf\_active, ukf\_active, ultrasonic\_active, imu\_mesh\_active$
end procedure

Algorithm 2 Convert Decision Tree to Graph

procedure DecisionTreeToGraph(DecisionTree)
    Initialize empty graph G
    Initialize node queue Q
    Enqueue root of DecisionTree to Q
    while Q is not empty **do**
        $current\_node$ = Dequeue Q
        Add $current\_node$ to G if not already present
        for each $child$ of $current\_node$ in DecisionTree **do**
            Add edge from $current\_node$ to $child$ in G
            Enqueue $child$ to Q
        end for
    end while
    return G
end procedure

Appendix C. Neural Network Details

GAT Architecture

The GAT model comprises two Graph Attention layers followed by a linear layer. The operations performed by each layer are:

First GAT Layer:

$$\mathbf{h}_{1i} = \text{ReLU}\left(\sum_{j \in \text{Neighbors}(i)} \alpha_{ij}^{(1)} \mathbf{W}_1 \mathbf{x}_j\right) \tag{45}$$

where $\mathbf{h}_{1i}$ is the hidden state of node i after the first GAT layer, $\alpha_{ij}^{(1)}$ is the attention coefficient between nodes i and j for the first GAT layer, $\mathbf{W}_1$ is the learnable weight for the first GAT layer, and $\mathbf{x}_j$ is the feature vector of node j. Afterward, a ReLU activation function is applied to $\mathbf{h}_1$.

Second GAT Layer:

$$\mathbf{h}_{2i} = \text{ReLU}\left(\sum_{j \in \text{Neighbors}(i)} \alpha_{ij}^{(2)} \mathbf{W}_2 \mathbf{h}_{1j}\right) \tag{46}$$

where $\mathbf{h}_{2i}$ is the hidden state of node i after the second GAT layer, $\alpha_{ij}^{(2)}$ is the attention coefficient between nodes i and j for the second GAT layer, $\mathbf{W}_2$ is the learnable weight for the second GAT layer, and $\mathbf{h}_{1j}$ is the hidden state of node j after the first GAT layer. Afterward, a ReLU activation function is applied to $\mathbf{h}_2$.

Global Mean Pooling:

$$\mathbf{g} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{h}_{2i} \tag{47}$$

where $\mathbf{g}$ is the graph-level feature vector, N is the total number of nodes, and $\mathbf{h}_{2i}$ is the hidden state of node i after the second GAT layer. Next, dropout is applied to the graph-level features $\mathbf{g}$.

Linear Layer:

$$\mathbf{o} = \mathbf{W}_{\text{lin}} \mathbf{g} \tag{48}$$

where $\mathbf{o}$ is the output vector representing class probabilities and $\mathbf{W}_{\text{lin}}$ is the learnable weight for the linear layer.

The GAT model is optimized using the Adam algorithm with a learning rate of 0.001 and a batch size of 32. The loss function employed is the weighted cross-entropy to handle class imbalance.

Matching Network

This section describes the Matching Network model, utilizing the graph embeddings inherited from the Graph Attention Network (GAT) to perform few-shot learning. The procedure is mathematically defined as follows:

Functions and Variables

- Create Support Query Sets: Generates a support set and a query set.

$$\text{support\_set} = \{\text{embeddings}[:N], \text{labels}[:N]\} \tag{49}$$

$$\text{query\_set} = \{\text{embeddings}[N:N+M], \text{labels}[N:N+M]\} \tag{50}$$

  Where N is the size of the support set, and M is the size of the query set.
- Compute Attention Weights: Computes the attention weights.

$$\text{attention\_weights} = \frac{\exp(\text{similarities})}{\sum \exp(\text{similarities})} \tag{51}$$

  Where similarities is the cosine similarity between query and support embeddings.
- Predict with Attention: Predicts the labels of the query set.

$$\text{weighted\_sum} = \text{attention\_weights} \times \text{support\_labels} \tag{52}$$

$$\text{predicted\_labels} = \begin{cases} 1 & \text{if weighted\_sum} > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{53}$$

Procedure

(1) The support and query sets are extracted via GAT graph embeddings.
(2) Cosine similarities are computed between embeddings in the query/support set.
(3) The attention weights are calculated via softmax function on the similarities.
(4) The weighted sum of the support labels is computed using the attention weights.
(5) A threshold of 0.5 is applied to predict binary labels for the query set.
(6) The mean accuracy of the predicted labels is calculated for evaluation.

Siamese Network

The section describes the training details for the Siamese Network. The Siamese Network aims to minimize the distance between similar items and maximize the distance between dissimilar items in the embedded space. The forward pass is given by:

$$\text{Forward Pass:} \quad z_1, z_2 = f_\theta(x_1), f_\theta(x_2) \tag{54}$$

Where f_θ is the network with param θ, x_1, x_2 are inputs, and z_1, z_2 are embeddings.

Contrastive Loss

The Contrastive Loss function is defined as:

$$L(y, z_1, z_2) = (1 - y) \times ||z_1 - z_2||^2 + y \times \max(0, m - ||z_1 - z_2||)^2 \tag{55}$$

Where y is the label (1:similar, 0:dissimilar), z_1, z_2 are embeddings, and m is margin.

Triplet Loss Function

The Triplet Loss function aims to ensure that an anchor point A in the feature space is closer to a positive point P than to a negative point N by a margin α. The loss is:

$$\text{TripletLoss}(A, P, N) = \max\left(\|f(A) - f(P)\|_2^2 - \|f(A) - f(N)\|_2^2 + \alpha, 0\right) \tag{56}$$

Where $f(x)$ is the feature transformation (or embedding) of x, $\|\cdot\|_2$ is the L_2 norm, A is the anchor point, P is the positive point, and N is the negative point.

K-Fold Cross-Validation

K-Fold Cross-Validation is used for model performance assessment. The data is divided into k subsets, and the model is trained on $k-1$ subsets and validated on the remainder.

Average accuracy and F1 Score are calculated as follows:

$$\text{Average Accuracy} = \frac{1}{k} \sum_{i=1}^{k} \text{Accuracy}_i \tag{57}$$

$$\text{Average F1 Score} = \frac{1}{k} \sum_{i=1}^{k} \text{F1 Score}_i \tag{58}$$