Emerson Hall

Ramana M. Pidaparti

2 September 2025

Data Preprocessing and Feature Engineering

Introduction

The goal of this week's assignment was to prepare a dataset for training the machine learning model designed in Week 5. Proper preprocessing is an essential step in machine learning because raw data often contains inconsistencies, redundant information or features that exist on very different scales. By preprocessing the UAV trajectory data generated by the A* baseline planner, the dataset becomes suitable for model training and evaluation. This week's work focused on normalizing the data which augments it to increase diversity, splitting it into training, validation and test sets, and extracting features that capture meaningful aspects of UAV flight.

Data Preparation

The dataset was generated directly from the MATLAB simulation code developed in Week 4. Each point along the planned trajectory was logged with position, velocity, goal information, and environmental measurements. A simple wind vector was also included to simulate external conditions. For each point, the code constructed a feature vector consisting of relative goal position, velocity components, wind conditions, and radial distances to nearby obstacles measured in eight compass directions. The corresponding labels were the velocity actions taken at each step along the path.

The raw features were standardized using z-score normalization. This process transformed each feature to have zero mean and unit variance, which ensures that no single feature dominates the training process because of scale differences. The normalized dataset was then divided into three subsets. Seventy percent of the samples were assigned to the training set, fifteen percent to the validation set, and the remaining fifteen percent to the test set. These splits guarantee that model evaluation is based on unseen data, preventing overfitting and providing a fair measure of generalization.

Feature Engineering

The feature engineering process was guided by the needs of trajectory optimization. The relative goal position provides global guidance toward the target location. The UAV velocity components capture the current motion state, which is necessary for continuous control. The wind vector adds environmental variability, preparing the model to handle realistic disturbances. Finally, the radial obstacle distances provide local spatial awareness by indicating how far the UAV can safely travel in each direction before encountering an obstacle.

By combining global, local, and environmental information, the feature set provides a comprehensive description of the state space. This design ensures that the learning algorithm has access to all relevant factors when choosing actions. The use of z-score normalization further stabilizes the learning process and reduces sensitivity to variations in raw data scales.

Results

The preprocessing process produced a complete dataset that is ready for machine learning experiments. The dataset contained 97 samples with 16 features each. Of these samples, 68 were placed in the training set, 15 in the validation set, and 14 in the test set. The mean value of the normalized features was close to zero across all splits, with the training set mean at 0.013 and standard deviation 0.738, the validation set mean at 0.046 and standard deviation 0.658, and the test set mean at −0.113 and standard deviation 0.797. These values confirm that normalization was applied correctly and that the distributions are balanced across splits.

The inclusion of obstacle distances and wind conditions increases the representativeness of the dataset and prepares the model for more challenging environments. The splits ensure that the model can be trained, tuned, and evaluated fairly, providing confidence that future performance measures will reflect real generalization ability.

Conclusion

In conclusion, the work completed in Week 6 successfully transformed raw trajectory outputs from the A* baseline planner into a normalized, feature-rich dataset suitable for machine learning. The data was augmented with wind conditions and obstacle distance features, standardized to a consistent scale, and divided into training, validation, and test sets. This dataset will serve as the foundation for training the Proximal Policy Optimization model designed in Week 5 and for evaluating its performance against the classical baseline established in Week 4.

Appendix

A) MATLAB Code

```
%% Week 6 logging, features, splits

clear; clc; close all;

% -------------------- Parameters --------------------
gridRes      = 0.20;
safetyMargin = 0.25;
xyMin = [-10 -10];
xyMax = [ 10  10];

startXY = [-8, -8];
goalXY  = [10,  4];

% -------------------- Obstacles ---------------------
wall = [0.5 -6; 0.7 -6; 0.7 6; 0.5 6];
bldg = [4 4; 7 4; 7 7; 4 7];

inflateRect = @(poly,r) [ min(poly(:,1))-r, min(poly(:,2))-r;
                          max(poly(:,1))+r, min(poly(:,2))-r;
                          max(poly(:,1))+r, max(poly(:,2))+r;
```

```
                    min(poly(:,1))-r, max(poly(:,2))+r ];
wallInf = inflateRect(wall, safetyMargin);
bldgInf = inflateRect(bldg, safetyMargin);

% ----------------- Occupancy Grid -------------------
res = gridRes;
xv = xyMin(1):res:xyMax(1);
yv = xyMin(2):res:xyMax(2);
[XC, YC] = meshgrid(xv, yv);
occ = inpolygon(XC, YC, wallInf(:,1), wallInf(:,2)) | ...
    inpolygon(XC, YC, bldgInf(:,1), bldgInf(:,2));

mapRows = size(occ,1); mapCols = size(occ,2);

% -------------- Start/Goal to grid idx --------------
world2grid = @(xy) [ round((xy(2)-xyMin(2))/res)+1, ...
              round((xy(1)-xyMin(1))/res)+1 ];
grid2world = @(rc) [ xyMin(1) + (rc(:,2)-1)*res, ...
              xyMin(2) + (rc(:,1)-1)*res ];

sRC = world2grid(startXY); gRC = world2grid(goalXY);

% --------------------- A* Setup ---------------------
gScore = inf(mapRows, mapCols); gScore(sRC(1),sRC(2)) = 0;
fScore = inf(mapRows, mapCols);
heur = @(r,c) norm([r c] - gRC);
fScore(sRC(1),sRC(2)) = heur(sRC(1),sRC(2));

cameFrom = zeros(mapRows, mapCols, 2, 'int32');
openSet  = false(mapRows, mapCols);
openSet(sRC(1), sRC(2)) = true;

neighbors = [ -1 -1; -1 0; -1 1; 0 -1; 0 1; 1 -1; 1 0; 1 1 ];
stepCost  = [ sqrt(2); 1; sqrt(2); 1; 1; sqrt(2); 1; sqrt(2) ];

nodesExpanded = 0;

% --------------------- A* Loop ----------------------
while true
   ftmp = fScore; ftmp(~openSet) = inf;
   [minVal, idxLin] = min(ftmp(:));
   if isinf(minVal), error('No path found.'); end
   [r, c] = ind2sub(size(occ), idxLin);
   openSet(r,c) = false;

   if r == gRC(1) && c == gRC(2)
```

```
        pathRC = [r c];
        while ~(r == sRC(1) && c == sRC(2))
            prev = double(squeeze(cameFrom(r,c,:))');
            if all(prev==0), break; end
            r = prev(1); c = prev(2);
            pathRC(end+1,:) = [r c]; %#ok<AGROW>
        end
        pathRC = flipud(pathRC);
        break;
    end

    nodesExpanded = nodesExpanded + 1;

    for k = 1:8
        rr = r + neighbors(k,1);
        cc = c + neighbors(k,2);
        if rr<1 || rr>mapRows || cc<1 || cc>mapCols, continue; end
        if occ(rr,cc), continue; end
        tentative = gScore(r,c) + stepCost(k);
        if tentative < gScore(rr,cc)
            cameFrom(rr,cc,:) = int32([r c]);
            gScore(rr,cc) = tentative;
            fScore(rr,cc) = tentative + heur(rr,cc);
            openSet(rr,cc) = true;
        end
    end
end

% ------------------- Path World ---------------------
pathXY = grid2world(pathRC);

% --------------------- Metrics ----------------------
seg = diff(pathXY,1,1);
pathLen = sum( sqrt(sum(seg.^2,2)) );
fprintf('Nodes expanded: %d\n', nodesExpanded);
fprintf('Path length (m): %.2f\n', pathLen);

% --------------------- Plot -------------------------
figure; hold on; axis equal;
patch(wallInf(:,1), wallInf(:,2), 'b', 'FaceAlpha',0.5, 'EdgeColor','k');   % wall blue
patch(bldgInf(:,1), bldgInf(:,2), [0.5 0.5 0.5], 'FaceAlpha',0.7, 'EdgeColor','k'); % block
grey
plot(startXY(1), startXY(2), 'go', 'MarkerFaceColor','g');
plot(goalXY(1),  goalXY(2),  'rx', 'LineWidth',2, 'MarkerSize',10);
plot(pathXY(:,1), pathXY(:,2), 'r-', 'LineWidth', 2);
xlabel('X (m)'); ylabel('Y (m)');
```

```
title('Baseline A* with Wall (Blue) and Block (Grey)');
grid on;

%% ==================== WEEK 6 ADDITIONS ====================
% Build a feature dataset from the path, normalize, split, and save.
% No anonymous "isFree" needed; all checks are explicit to avoid syntax issues.

% --- Settings for feature engineering ---
flightAlt = 0.0;             % using 2D path here
wind_xy   = [0.2, -0.1];      % simple constant wind feature
dirs_deg  = 0:45:315;         % 8 directions
dirs_rad  = deg2rad(dirs_deg)';
ray_max   = 5.0;              % meters
ray_step  = res * 0.5;        % sampling step along each ray

% --- Compute velocities along the path (finite difference) ---
N = size(pathXY,1);
velXY = zeros(N,2);
if N >= 2
   velXY(1:end-1,:) = diff(pathXY,1,1);
   velXY(end,:)      = velXY(end-1,:);
end

% --- Allocate logs ---
% Features per point: [rel_goal_x, rel_goal_y, rel_goal_z(=0), vx, vy, vz(=0), wind_x,
wind_y, 8*ray_dists]
log_feats = zeros(N, 3 + 3 + 2 + 8);
log_act   = zeros(N, 3);              % [vx vy vz], z=0
log_done  = zeros(N, 1);
log_pos   = [pathXY, zeros(N,1)];     % [x y z], z=0
log_goal  = repmat([goalXY, flightAlt], N, 1);
log_vel   = [velXY, zeros(N,1)];
log_wind  = repmat(wind_xy, N, 1);

% --- Radial distances and feature assembly per point ---
for i = 1:N
   p = pathXY(i,:);                      % [x y]
   rel_goal = [goalXY - p, 0.0];          % z diff is 0
   vxy = velXY(i,:);

   % Cast 8 rays and measure distance to first obstacle or boundary
   dist8 = zeros(8,1);
   for k = 1:numel(dirs_rad)
      theta = dirs_rad(k);
      dir = [cos(theta), sin(theta)];
      hit = ray_max;
```

```
        % sample along ray
        s = 0.0;
        while s <= ray_max
            q = p + s*dir;                  % world point
            % map to grid indices
            rr = round((q(2)-xyMin(2))/res) + 1;   % row (y)
            cc = round((q(1)-xyMin(1))/res) + 1;   % col (x)
            % out-of-bounds => treat as obstacle
            if rr < 1 || rr > mapRows || cc < 1 || cc > mapCols
                hit = s; break;
            end
            % occupied cell => obstacle
            if occ(rr,cc)
                hit = s; break;
            end
            s = s + ray_step;
        end
        dist8(k) = hit;
    end

    % Assemble feature vector and labels
    feat = [rel_goal, vxy, 0.0, wind_xy, dist8'];
    log_feats(i,:) = feat;
    log_act(i,:)   = [vxy, 0.0];
    log_done(i)    = double(i == N);
end

% --- Normalize features (z-score) ---
mu = mean(log_feats, 1);
sigma = std(log_feats, 0, 1); sigma(sigma==0) = 1;
feats_norm = (log_feats - mu) ./ sigma;

% --- Train/Val/Test split ---
M = size(feats_norm,1);
rng(1);
perm = randperm(M);
nTrain = round(0.7*M);
nVal   = round(0.15*M);
train_idx = perm(1:nTrain);
val_idx   = perm(nTrain+1 : nTrain+nVal);
test_idx  = perm(nTrain+nVal+1 : end);

% --- Save dataset ---
dataset = struct();
dataset.features    = feats_norm;
dataset.actions     = log_act;
```

```
dataset.done        = log_done;
dataset.mu          = mu;
dataset.sigma       = sigma;
dataset.wind_xy     = log_wind;
dataset.pos         = log_pos;
dataset.vel         = log_vel;
dataset.goal        = log_goal;
dataset.split.train = train_idx;
dataset.split.val   = val_idx;
dataset.split.test  = test_idx;

save('week6_dataset.mat', '-struct', 'dataset');

% Also CSVs for quick inspection
writematrix(feats_norm(train_idx,:), 'week6_features_train.csv');
writematrix(feats_norm(val_idx,:),   'week6_features_val.csv');
writematrix(feats_norm(test_idx,:),  'week6_features_test.csv');
writematrix(log_act,                 'week6_actions.csv');
writematrix(log_done,                'week6_done.csv');

disp('Week 6 dataset saved: week6_dataset.mat and CSV files.');
```