Autonomous Navigation.py > ...

```python
import heapq
def a_star(grid, start, goal):
    def heuristic(a, b):
        return abs(a[0] - b[0]) + abs(a[1] - b[1])
    open_set = []
    heapq.heappush(open_set, (0 + heuristic(start, goal), 0, start, [start]))
    visited = set()
    while open_set:
        _, cost, current, path = heapq.heappop(open_set)
        if current in visited:
            continue
        if current == goal:
            return path
        visited.add(current)
        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
            neighbor = (current[0] + dx, current[1] + dy)
            if (0 <= neighbor[0] < len(grid) and 0 <= neighbor[1] < len(grid[0])
                    and grid[neighbor[0]][neighbor[1]] == 0):
                heapq.heappush(open_set, (cost + 1 + heuristic(neighbor, goal),
                                          cost + 1, neighbor, path + [neighbor]))
    return None
grid = [[0, 0, 0, 1],
    [0, 1, 0, 0],
    [0, 0, 0, 0],]
start = (0, 0)
goal = (2, 3)
path = a_star(grid, start, goal)
print("Path found:", path)
```

Robotic Control System.py > ...

```python
class PIDController:
    def __init__(self, kp, ki, kd):
        self.kp, self.ki, self.kd = kp, ki, kd
        self.prev_error = 0
        self.integral = 0

    def compute(self, target, current):
        error = target - current
        self.integral += error
        derivative = error - self.prev_error
        self.prev_error = error
        return self.kp * error + self.ki * self.integral + self.kd * derivative

# Example usage
pid = PIDController(kp=1.0, ki=0.1, kd=0.05)
speed = 0
target_speed = 10
for i in range(10):
    control = pid.compute(target_speed, speed)
    speed += control * 0.1  # Simulated actuator response
    print(f"Step {i}: Speed = {speed:.2f}")
```

Sensor Fusion.py > KalmanFilter

```python
class KalmanFilter:
    def __init__(self):
        self.x = 0      # Position
        self.P = 1      # Estimation uncertainty
        self.Q = 0.1    # Process variance
        self.R = 0.5    # Measurement variance

    def update(self, z):
        # Prediction step
        self.P = self.P + self.Q

        # Update step
        K = self.P / (self.P + self.R)
        self.x = self.x + K * (z - self.x)
        self.P = (1 - K) * self.P
        return self.x

# Example usage
kf = KalmanFilter()
measurements = [5.0, 5.5, 6.0, 5.8, 6.2]
for z in measurements:
    estimate = kf.update(z)
    print(f"Measurement: {z:.2f} => Estimated Position: {estimate:.2f}")
```