```
diagram SwarmTaxisController

type int
type bool
type real

interface SwarmTaxisIface {
    Turn() : void
    CalcCoherenceHeading() : void
    CalcAvoidanceHeading() : void
    CheckIlluminationStatus(): bool
    UpdateAvoidanceRadius(): void

    RotateClockwise(i:real) : void
    RotateAntiClockwise(i:real) : void
    MoveForward(i: real) : void

    event anyRobotToAvoid
    var linearSpeed : real
    var angularSpeed : real
    var reached : bool
    var desiredTurningDegree: real
    var degreeTurned : real
    var avoidanceRadius : real
}


stm SwarmTaxisFSM {
    requires SwarmTaxisIface

    clock discrete T
    initial Initial

    state Forward {
        entry reached = false; UpdateAvoidanceRadius(); MoveForward(6.4)
    }
    state Coherence {
        entry CalcCoherenceHeading(); Turn()
    }
    state Avoidance {
        entry CalcAvoidanceHeading(); Turn()
    }

    transition t1 {
        from Initial
        to Forward
        trigger #T
    }
    transition t2 {
        from Forward
        to Coherence
        condition since(T) > 25
    }
    transition t3 {
        from Forward
        to Avoidance
        trigger anyRobotToAvoid
    }

    transition t4 {
        from Coherence
        to Forward
        trigger #T
        condition reached == true
    }

    transition t5 {
        from Avoidance
        to Forward
        trigger #T
        condition reached == true
    }
}


operation Turn() : void {

    requires SwarmTaxisIface

    initial I
    final F
    state S {entry if desiredTurningDegree <= 0 then RotateClockwise(-2.51)
                   else RotateAntiClockwise(2.51) end;
                   degreeTurned = degreeTurned + angularSpeed * 0.1;
                   if (desiredTurningDegree < 0 /\ degreeTurned <
desiredTurningDegree) then reached = true; degreeTurned = 0 else skip end;
                   if (desiredTurningDegree > 0 /\ degreeTurned >
desiredTurningDegree) then reached = true; degreeTurned = 0 else skip end
    }
    transition t1 {
        from I
        to S
    }
    transition t2 {
        from S
        to F
    }
}

operation UpdateAvoidanceRadius(): void {

    var illuminated: bool
    requires SwarmTaxisIface

    initial I
    final F
    state S {
            entry illuminated = CheckIlluminationStatus(); if illuminated ==
true then avoidanceRadius = 0.2
                  else avoidanceRadius = 0.1 end
    }
    transition t1 {
        from I
        to S
    }
    transition t2 {
        from S
        to F
    }
}

operation MoveForward(i : real) : void {
    requires SwarmTaxisIface
    precondition i > 0

    initial I
    final F
    state s1 {entry linearSpeed = i; angularSpeed = 0
    }
    transition t1 {
        from I
        to s1
    }
    transition t2 {
        from s1
        to F
    }
}

operation RotateClockwise(j:real) : void {
    precondition j < 0

    requires SwarmTaxisIface
    initial I
    final F
    state s1 {entry linearSpeed = 0; angularSpeed = j
    }
    transition t1 {
        from I
        to s1
    }
    transition t2 {
        from s1
        to F
    }
}

operation RotateAntiClockwise(j:real) : void {
    precondition j > 0

    requires SwarmTaxisIface
    initial I
    final F
    state s1 {entry linearSpeed = 0; angularSpeed = j
    }
    transition t1 {
        from I
        to s1
    }
    transition t2 {
        from s1
        to F
    }
}
```