Supervisory Control Final Assignment

2PDASDSCS

Supervisory Control of a Warehouse Robot System

Authors:
A. Kalra
P. Biswas
T.S.R. Parvathaneni

December 17, 2019

1 Introduction

The project is based on Supervisory Control of a Warehouse Robot System provided as a final assignment for the course on Supervisory Control Synthesis (2PDASDSCS). This project focuses on connected automated guided vehicles (AGV's) in warehouses and distribution centers. The goal of the project is to design an architecture of a supervisory control system so that multiple AGV's can operate simultaneously in a warehouse. Within the scope of this project, a warehouse consists of four separate areas, storage, driving, picking and charging with multiple interacting AGV's as shown in Figure 1. Here each AGV needs to perform certain tasks (functions), while not hampering with the operation of other AGV's. Another main objective of the supervisor is to ensure safe movement of all the AGV's in the warehouse at all times.

In this report Section 2 details the assumptions made in order to simplify the project. Next, Section 3 discusses the models of different plants involved in the warehouse system. Furthermore, Section 4 describes the requirements considered to design the supervisor. Finally, Section 5 presents the supervisor synthesized by considering the assumptions, the plant models and the requirements.

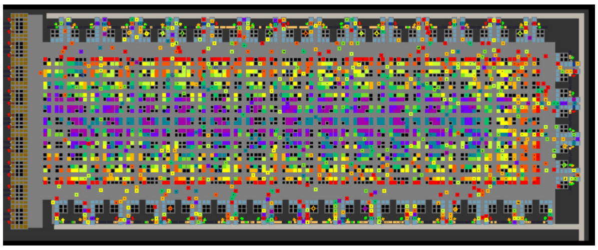

Figure 1: Simulation environment of the Amazon Robotics system used to design and validate the system [1]

2 Assumptions

The design of a real implementable supervisory control system for a warehouse requires considering multiple aspects concerning each and every movement of the AGV's comprehensively. Since the aim of the of the project is mainly to introduce us to the CIF simulation environment, modelling of a plant, formulation of the requirements and not to design a full implementable supervisor, therefore we make certain assumptions. These assumptions help us simplify the problem, hence making it tractable. The assumptions that we have considered in our project are as follows:

1. It is assumed that human intervenes after a set time period when an obstacle is encountered and the AGV is not stuck in a loop indefinitely.

2. A total of two AGVs are considered in the warehouse with respect to the CIF implementation, although the synthesized supervisor is generalized with respect to m AGVs, where $m \in \mathbb{N}$.

3. The charging stations are located in initial position of the AGVs and a job can be allocated to the AGV only when it is in it's initial position. The AGV can indicate completion of a job only when it reaches the initial position again and then be in the idle or available state.

4. The location camera sensor, IR proximity sensor and the touch sensitive sensor have been assumed to be powered and functional through the entire job cycle, including when the AGV is at the charging station.

5. It is assumed that the rotation angle for the AGV at any point of time is evaluated by the path planner and then communicated to the supervisor. Thereafter, the supervisor is responsible for rotating the AGV. Once the rotation command is executed the control of the AGV is passed back to the path planning controller.

6. It is assumed that all the low level controllers, communication systems, actuators and sensors never fail and are fully operational at all times.

7. It assumed that the AGV always aligns properly with the product stack hence eliminating the need to model the product stack barcode camera, and thus the corresponding requirements.

3 Plant Models

Plant models refer to any uncontrolled system within the warehouse, which need to be controlled by the supervisor in order to adhere to specific requirements such as safety. It should be noted that plant models represent the natural behavior of any system. Thus, we do not restrict the fundamental behavior of the plant in our work. Below, we list and discuss all the plant models considered in this project.

a. **Vehicle Movement:** This models the forward movement of the AGV as shown in Figure 2. This is done by using two controllable events, *start* and *stop*, assuming AGV is not moving initially i.e. AGV is in the idle state. Once the supervisor triggers the *start* event the AGV state changes to *move*, while the *stop* event changes AGV state back to *idle*.

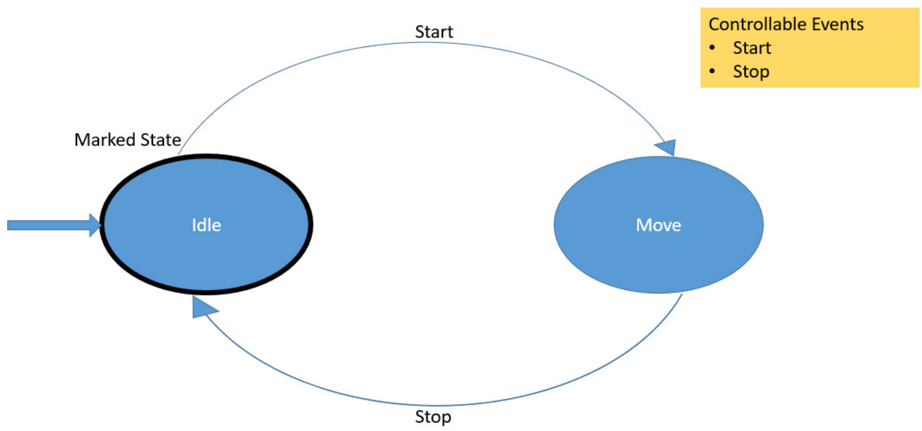

Figure 2: Vehicle Movement Plant

b. **Vehicle Auxiliary Movement:** Auxiliary movement of the AGV relates to the rotation mechanism. The AGV is allowed to rotate by a certain angle as commanded by the supervisor. This is achieved by defining two controllable events *rotate* and *stop rotate*, as shown in Figure 3. The supervisor

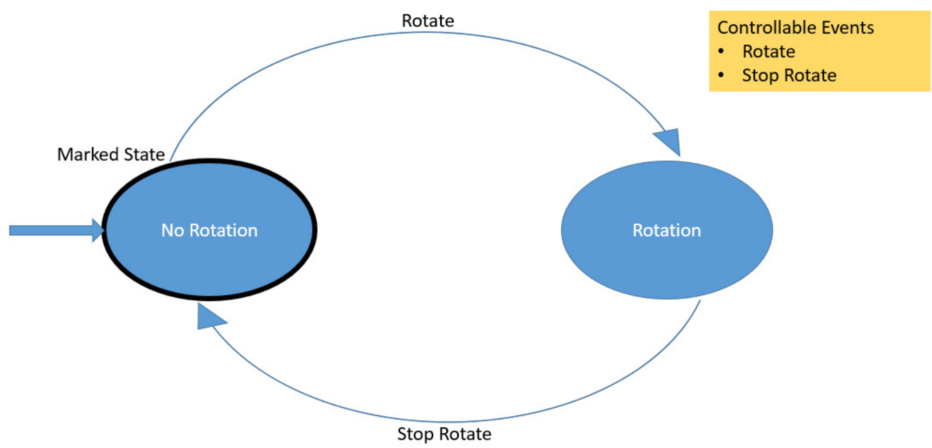

Figure 3: Vehicle Auxiliary Movements Plant

can trigger the event *rotate* to start rotating the AGV, while *stop rotate* event enables the supervisor to stop the rotation of the AGV.

c. **Vehicle Status:** The status of the AGV is tracked using five states, *idle, available, fetching destination, in execution* and *busy* as shown in Figure 4. This AGV starts from the *idle* state initially. The supervisor can trigger the *is available* event to change the state of the AGV to *available*. Next, the event *get destination* allows the supervisor to start fetching the destination from the resource allocation controller. Once the supervisor has acquired the destination, it can communicate the current location and the fetched destination to the path planner, which starts the execution of the path planner. In this context execution refers to enabling the path planner to start planning the path and not execution of the path by the AGV. Once the path is planned, the supervisor can use the event *in process* to allow the AGV to move on the planned path. Finally, when the AGV reaches the fetched destination, the state of AGV is changed to *idle* using the *job completed* event.

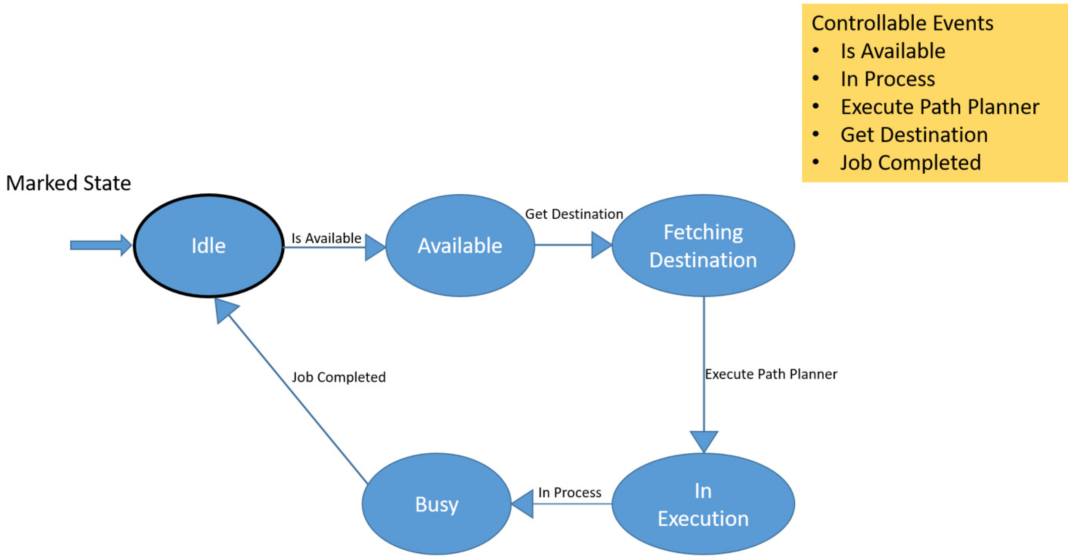

Figure 4: Vehicle Status Plant

d. **Stack Lifting Mechanism:** The stack lifting mechanism plant models the ability of the AGV to raise and lower the lever. The AGV initially starts from a *low* position, where *raise* event enables

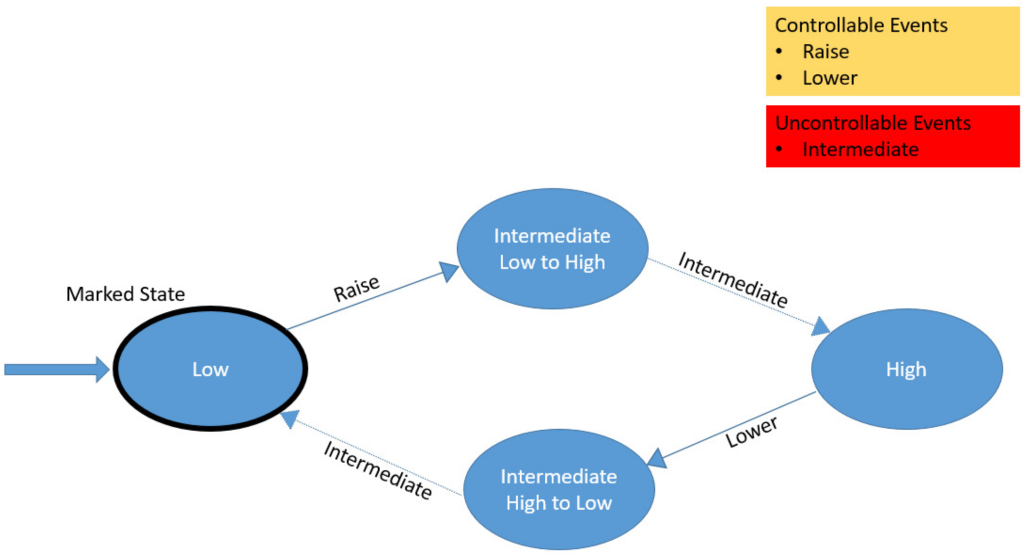

Figure 5: Stack Lifting Mechanism Plant

the supervisor to raise the lever of the AGV, while *lower* event allows the AGV to bring down the lever, as shown in Figure 5. The AGV uses sensors, one at the high position and another at the low position. Whenever the high sensor is true the AGV considers the lever to be raised. On the other hand, when the low sensor is true the AGV considers the lever to be at the low position. When none of the sensors are true the AGV considers the lever to be somewhere in between the low and high positions, denoted by the intermediate states. In order to enable the supervisor to understand whether the lever is moving from low to high or high to low, two intermediate states are introduced, sharing the same uncontrollable event *intermediate*.

e. **Touch and Proximity Sensor:** Touch and proximity sensors have the same plant model design as shown in Figure 6. Here, touch sensor is responsible for detecting contact with any obstacle, while the proximity sensor detects the presence of an obstacle in the front and rear directions of the AGV, upto a certain distance. In case of both the sensors the initial state is considered to be *off*. The plant here consists of two uncontrollable events *turning on* and *turning off*. The events are considered to be uncontrollable as the supervisor can't control when a obstacle will collide with the AGV. Therefore, whenever the event *turning on* is triggered the supervisor changes the state of the sensors to *on*. As soon as the obstacle goes away which is denoted by event *turning off*, the supervisor changes the state of the sensor to *off*. The states *on* and *off* here represent whether the sensor is detecting any obstacle or not, respectively. It does not signify whether the sensor itself is on or off, instead the sensors are considered to be on and operational all the time.

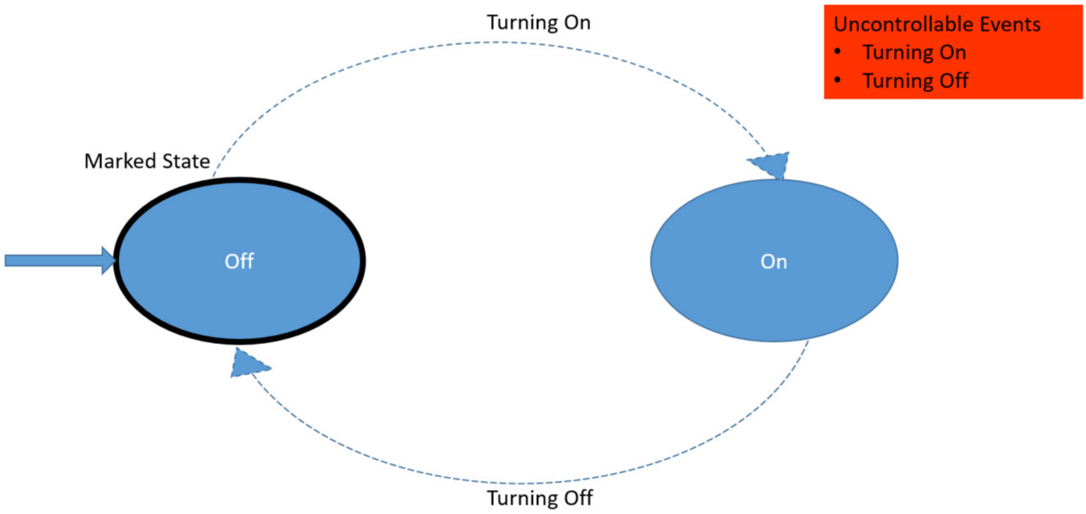

Figure 6: Proximity Sensor Plant

f. **Location Camera Sensor:** The location camera sensor is used to scan the barcodes on the grid to enable the AGV to identify it's location. This is done by modelling the plant using two states *on* and *off*, as shown in Figure 7. The states *on* and *off* do not represent whether the camera is on or not, in fact camera is on all the time. Instead the states just represent two locations without any physical significance. The plant also has two events, a controllable event *scanning* and an uncontrollable event *detected*. *Detected* is considered a uncontrollable event as the supervisor cannot control as to when a barcode will be detected by the sensor. When ever the *detected* event occurs the state changes to *on*, while triggering of the *scanning* event represents that the camera is scanning the surrounding area.

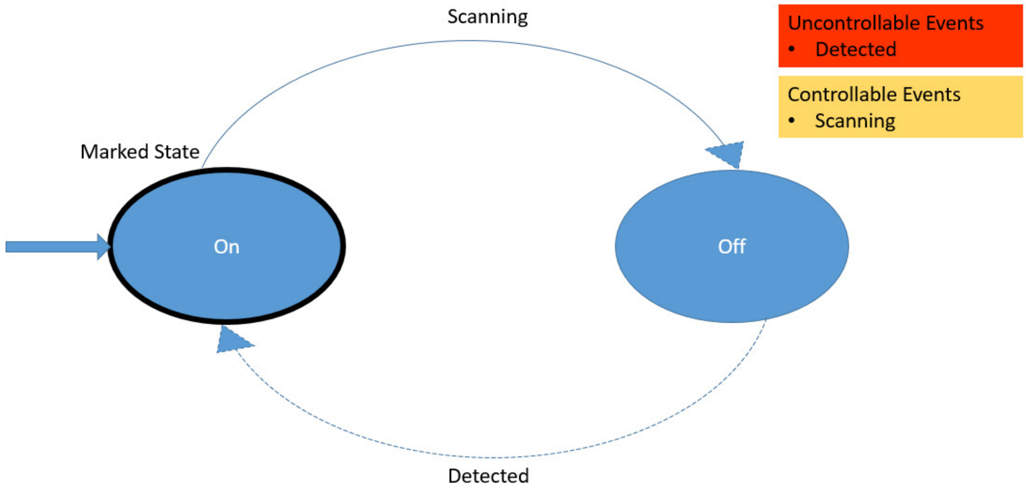

Figure 7: Location Camera Sensor Plant

g. **Battery Charging:** The battery charging plant automaton is modelled using three states, *idle*, *fetch*, *charging* and three controllable events, *fetch charging station*, *start charging* and *stop charging*, as shown in Figure 8. Initially the automaton is assumed to be in the *idle* state. Next, the *fetch charging station* event leads to the AGV asking for a charging location to the resource allocation controller. Once the charging location is fetched the AGV can move to the destination. After the AGV has reached the fetched charging destination, the supervisor can trigger the event *start charging* so that AGV can charge. Finally, the *stop charging* event stops charging the AGV and brings the automaton back to the *idle* state.

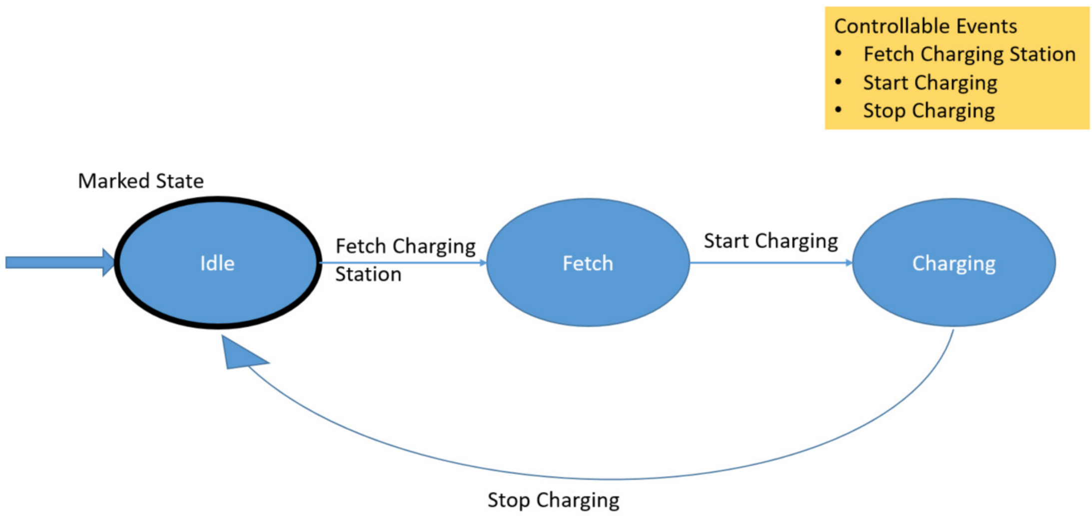

Figure 8: Battery Charging Plant

h. **Battery Level:** The battery level plant monitors the level of the battery that AGV has at any point of time. This is done by using three states, *full*, *low* and *critical*, with *full* being the initial state as shown in Figure 9. The plant also consists of three uncontrollable events *increase*, *decrease* and *replace*. The event *decrease* changes the state of the battery from *full* to *low* and *low* to *critical* depending upon the instantaneous state of the battery at any point of time. On the other hand *increase* event occurs when the battery starts charging resulting in state transforming to *full*. Once the battery is critical, it needs to be replaced by a human with a full battery which is again an uncontrollable with respect to the supervisor.

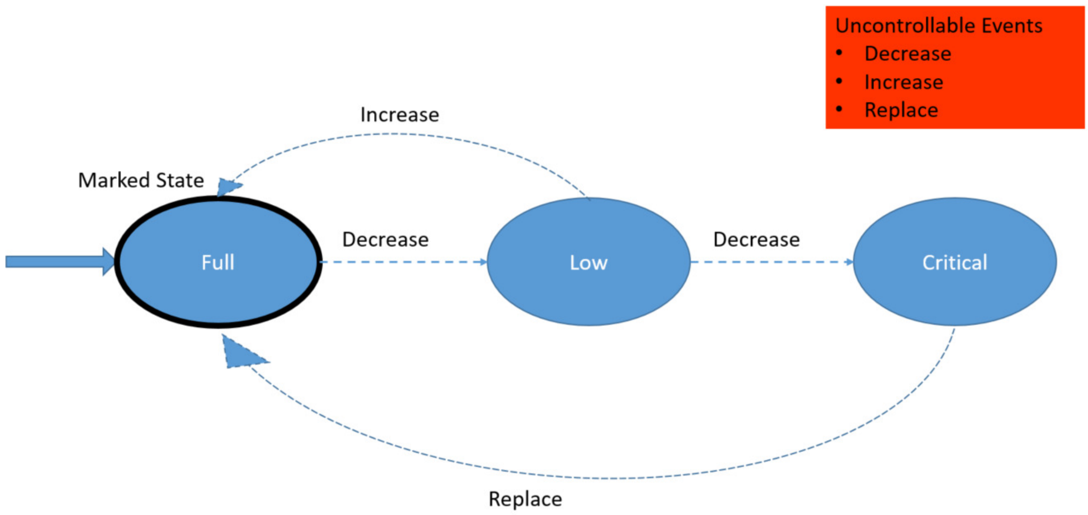

Figure 9: Battery Level Plant

4 Requirements

In order to restrict the supervisor from triggering events that are not desirable in certain conditions, a set of conditions can be imposed. These conditions allow to model functions that the whole system is allowed to perform. Such conditions are referred to as requirements. Here, we first explain the informal requirements with respect to each of the plants described in Section 3. Next, the formal requirements are explained.

4.1 Informal

a. **Vehicle Movement:** It is desired that while the AGV is moving, no rotation or stack lifting movements should be allowed. Also, whenever either the proximity sensor or the obstacle sensor gets activated, AGV should stop moving. One more requirement is that the AGV cannot start moving till the time it has received the destination from the resource allocation controller and also till the time the path has been planned by the path planning controller. Finally, if the battery level is critically low, the AGV cannot move.

b. **Vehicle Auxiliary Movements:** As explained earlier, the AGV rotation is not allowed whenever the vehicle is moving. Apart from that AGV cannot rotate even when the stack lifting mechanism is operating or when the AGV is charging at a charging location.

c. **Stack Lifting Mechanism:** The stack lifting mechanism is required not to operate whenever the AGV is moving, rotating or charging at the charging area.

d. **Proximity Sensor:** It is required that whenever the proximity sensor detects an obstacle the AGV should stop moving.

e. **Camera Sensor:** Similar to the proximity sensor requirement, here also it is required that whenever the touch sensor detects an obstacle the AGV should stop moving.

f. **Battery Charging:** Here, the main requirement is that whenever the battery level is critically low, the AGV should stop moving and wait for human assistance. If the battery is low but not critically, the AGV must go to the charging station to increase the battery level.

4.2 Formal

The requirements stated above are formally formulated here, partially using requirement automata and partially using requirement invariants. Initially the requirement automata are explained followed by the requirement invariants. In our work the following requirement automata are used:

1. **Path Scanning:** As implemented between line 114 - 121 in the cif code described in Section 5, this automata allows the rotation and stack lifting mechanism of the AGV to operate only when the location camera sensor plant is in the *on* state, i.e. the AGV has detected the barcode already and the movement has stopped.

2. **Charge when battery is low:** As stated between lines 122 - 131, this requirement automaton specifies the following conditions:

   - The AGV should stop charging as soon as the battery level is full by jumping to the *no charging* state.
   - If the battery level becomes low, the AGV is supposed to fetch location of the charging station by changing it's state to *getting location*.
   - As soon as the AGV receives the charging location, AGV needs to move towards the charging location. When it reaches the charging location the AGV can start charging given that it is not rotating, stack lifting mechanism is not operating and the battery level is not critically low.

3. **Stop all movements when charging:** This requirement between lines 132 - 141, the following conditions are stated by this requirement automaton:

   - The AGV can only move when the touch and the proximity sensors are not on, battery level is full, AGV is in the *busy* state or when the battery level is low with the touch and the proximity sensors in the *off* state.
   - Next, the AGV can use the stack lifting mechanism only when AGV movement is in the *idle* state, with it's status being *busy* while no rotation is happening.
   - Thirdly, the AGV can rotate only when it is not moving, with the stack not being in an intermediate position. The AGV can move when the battery level is low and not when it is critically low.
   - Finally, AGV has to stop if either the touch or obstacle sensor get *on*, or battery level becomes critically low or a barcode is detected.

Apart form these requirement automata, the following event disabling requirements are implemented using requirement invariants:

1. Line 294 defines an invariant requirement stating AGV stack lifting mechanism cannot raise if battery is low or critically low.

2. Line 295 defines an invariant requirement stating AGV cannot rotate if battery is critically low.

3. Line 296 - 299 defines invariant requirements stating that whenever battery is low or critically low, the AGV is not available to perform any job except for charging itself and moving to the charging location.

4. Line 300 defines an invariant requirement stating the job cannot be complemented if the AGV is rotating, or stack lifting mechanism is operating, or final destination bar code is not detected or the battery level gets critically low.

5. Line 301 defines an invariant requirement stating AGV cannot rotate when it is moving.

6. Line 302 defines an invariant requirement stating AGV cannot start moving when it is rotating.

7. Line 303 - 304 defines invariant requirements stating that AGV cannot move when the stack lifting mechanism is at an intermediate position.

5 Synthesized CIF Code

```
group warehouse1:
  group robot1:
    plant automaton VehicleMovement:
      controllable c_start;
      controllable c_stop;
      location idle:
        initial;
        marked;
        edge c_start goto move;
      location move:
        edge c_stop goto idle;
    end
    plant automaton AuxiliaryMovements:
      controllable c_rotate;
      controllable c_stop, c_rotate;
      location NoRotation:
        initial;
        marked;
        edge c_rotate goto Rotation;
      location Rotation:
        edge c_stopc_rotate goto NoRotation;
    end
    plant automaton VehicleLever:
      controllable c_Raise;
      controllable c_Lower;
      uncontrollable u_intermediate;
      location low:
        initial;
        marked;
        edge c_Raise goto intermediate_LowToHigh;
      location intermediate_LowToHigh:
        edge u_intermediate when VehicleStatus.Busy = true and BatteryLevel.low =
            false and BatteryLevel.critical = false and AuxiliaryMovements.Rotation =
            false goto high;
      location intermediate_HighToLow:
        edge u_intermediate when VehicleStatus.Busy = true and BatteryLevel.low =
            false and BatteryLevel.critical = false and AuxiliaryMovements.Rotation =
            false goto low;
      location high:
        edge c_Lower goto intermediate_HighToLow;
    end
    plant automaton ObstacleSensor:
      uncontrollable u_TurningOn;
      uncontrollable u_TurningOff;
      location off:
        initial;
        marked;
        edge u_TurningOn goto on;
      location on:
        edge u_TurningOff goto off;
    end
    plant automaton TouchSensor:
      uncontrollable u_TurningOn;
      uncontrollable u_TurningOff;
      location off:
        initial;
        marked;
        edge u_TurningOn goto on;
```

```
    location on:
      edge u_TurningOff goto off;
  end
  plant automaton BatteryLevel:
    uncontrollable u_Increase;
    uncontrollable u_Decrease;
    uncontrollable u_Replace;
    location full:
      initial;
      marked;
      edge u_Decrease when ChargingSystem.Charging = false goto low;
    location low:
      edge u_Increase when ChargingSystem.Charging = true goto full;
      edge u_Decrease when ChargingSystem.Charging = false goto critical;
    location critical:
      edge u_Replace goto full;
  end
  plant automaton VehicleStatus:
    controllable c_isAvailable;
    controllable c_inProcess;
    controllable c_ExecutePathPlanner;
    controllable c_GetDestination;
    controllable c_JobCompleted;
    location idle:
      initial;
      marked;
      edge c_isAvailable goto Available;
    location Available:
      edge c_GetDestination goto FetchingDestination;
    location FetchingDestination:
      edge c_ExecutePathPlanner goto inExecution;
    location inExecution:
      edge c_inProcess goto Busy;
    location Busy:
      edge c_JobCompleted goto idle;
  end
  plant automaton CameraSensor:
    uncontrollable u_Detected;
    controllable c_Scanning;
    location on:
      initial;
      marked;
      edge c_Scanning goto off;
    location off:
      edge u_Detected goto on;
  end
  plant automaton ChargingSystem:
    controllable c_startCharging;
    controllable c_stopCharging;
    controllable c_fetch_ChargingStation;
    location idle:
      initial;
      marked;
      edge c_fetch_ChargingStation goto fetch;
    location fetch:
      edge c_startCharging goto Charging;
    location Charging:
      edge c_stopCharging goto idle;
  end
  supervisor automaton PathScanning:
    location Deactive:
```

```
      initial;
      marked;
      edge AuxiliaryMovements.c_rotate when CameraSensor.on;
      edge VehicleLever.c_Raise when CameraSensor.on;
      edge VehicleLever.c_Lower when CameraSensor.on;
  end
  supervisor automaton ChargeWhenBatteryIsLow:
    location Getting_Location:
      edge ChargingSystem.c_startCharging when BatteryLevel.low and
          VehicleMovement.idle and AuxiliaryMovements.NoRotation and
          ObstacleSensor.off and TouchSensor.off and CameraSensor.on goto Charging;
    location NoCharging:
      initial;
      marked;
      edge ChargingSystem.c_fetch_ChargingStation when BatteryLevel.low goto
          Getting_Location;
    location Charging:
      edge ChargingSystem.c_stopCharging when BatteryLevel.full goto NoCharging;
  end
  supervisor automaton stopAllMovementsWhenCharging_Lever:
    location Deactive:
      initial;
      marked;
      edge VehicleMovement.c_start when ObstacleSensor.off and TouchSensor.off and
          BatteryLevel.full and ChargingSystem.idle and VehicleStatus.Busy or
          ObstacleSensor.off and TouchSensor.off and BatteryLevel.low and
          ChargingSystem.fetch and ChargeWhenBatteryIsLow.Getting_Location;
      edge VehicleLever.c_Raise when ChargingSystem.idle and
          AuxiliaryMovements.NoRotation and VehicleMovement.idle and
          VehicleStatus.Busy;
      edge VehicleLever.c_Lower when ChargingSystem.idle and
          AuxiliaryMovements.NoRotation and VehicleMovement.idle and
          VehicleStatus.Busy;
      edge AuxiliaryMovements.c_rotate when BatteryLevel.full and VehicleStatus.Busy
          and VehicleLever.intermediate_LowToHigh = false and ChargingSystem.idle
          and VehicleLever.intermediate_HighToLow = false or ChargingSystem.fetch
          and ChargeWhenBatteryIsLow.Getting_Location;
      edge VehicleMovement.c_stop when ObstacleSensor.on or TouchSensor.on or
          CameraSensor.on or BatteryLevel.critical;
  end
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleLever.c_Raise;
  requirement invariant BatteryLevel.critical = true disables
      AuxiliaryMovements.c_rotate;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_isAvailable;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_inProcess;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_ExecutePathPlanner;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_GetDestination;
  requirement invariant AuxiliaryMovements.Rotation or BatteryLevel.low = true or
      CameraSensor.off or VehicleLever.intermediate_LowToHigh or
      VehicleLever.intermediate_HighToLow or BatteryLevel.critical = true or
      ChargingSystem.Charging disables VehicleStatus.c_JobCompleted;
  requirement invariant VehicleMovement.move disables AuxiliaryMovements.c_rotate;
  requirement invariant AuxiliaryMovements.Rotation disables VehicleMovement.c_start;
  requirement invariant VehicleLever.intermediate_LowToHigh disables
      VehicleMovement.c_start;
```

```
  requirement invariant VehicleLever.intermediate_HighToLow disables VehicleMovement.c_start;
end
group robot2:
  plant automaton VehicleMovement:
    controllable c_start;
    controllable c_stop;
    location idle:
      initial;
      marked;
      edge c_start goto move;
    location move:
      edge c_stop goto idle;
  end
  plant automaton AuxiliaryMovements:
    controllable c_rotate;
    controllable c_stopc_rotate;
    location NoRotation:
      initial;
      marked;
      edge c_rotate goto Rotation;
    location Rotation:
      edge c_stopc_rotate goto NoRotation;
  end
  plant automaton VehicleLever:
    controllable c_Raise;
    controllable c_Lower;
    uncontrollable u_intermediate;
    location low:
      initial;
      marked;
      edge c_Raise goto intermediate_LowToHigh;
    location intermediate_LowToHigh:
      edge u_intermediate when VehicleStatus.Busy = true and BatteryLevel.low = false and BatteryLevel.critical = false and AuxiliaryMovements.Rotation = false goto high;
    location intermediate_HighToLow:
      edge u_intermediate when VehicleStatus.Busy = true and BatteryLevel.low = false and BatteryLevel.critical = false and AuxiliaryMovements.Rotation = false goto low;
    location high:
      edge c_Lower goto intermediate_HighToLow;
  end
  plant automaton ObstacleSensor:
    uncontrollable u_TurningOn;
    uncontrollable u_TurningOff;
    location off:
      initial;
      marked;
      edge u_TurningOn goto on;
    location on:
      edge u_TurningOff goto off;
  end
  plant automaton TouchSensor:
    uncontrollable u_TurningOn;
    uncontrollable u_TurningOff;
    location off:
      initial;
      marked;
      edge u_TurningOn goto on;
    location on:
```

```
      edge u_TurningOff goto off;
    end
    plant automaton BatteryLevel:
      uncontrollable u_Increase;
      uncontrollable u_Decrease;
      uncontrollable u_Replace;
      location full:
        initial;
        marked;
        edge u_Decrease when ChargingSystem.Charging = false goto low;
      location low:
        edge u_Increase when ChargingSystem.Charging = true goto full;
        edge u_Decrease when ChargingSystem.Charging = false goto critical;
      location critical:
        edge u_Replace goto full;
    end
    plant automaton VehicleStatus:
      controllable c_isAvailable;
      controllable c_inProcess;
      controllable c_ExecutePathPlanner;
      controllable c_GetDestination;
      controllable c_JobCompleted;
      location idle:
        initial;
        marked;
        edge c_isAvailable goto Available;
      location Available:
        edge c_GetDestination goto FetchingDestination;
      location FetchingDestination:
        edge c_ExecutePathPlanner goto inExecution;
      location inExecution:
        edge c_inProcess goto Busy;
      location Busy:
        edge c_JobCompleted goto idle;
    end
    plant automaton CameraSensor:
      uncontrollable u_Detected;
      controllable c_Scanning;
      location on:
        initial;
        marked;
        edge c_Scanning goto off;
      location off:
        edge u_Detected goto on;
    end
    plant automaton ChargingSystem:
      controllable c_startCharging;
      controllable c_stopCharging;
      controllable c_fetch_ChargingStation;
      location idle:
        initial;
        marked;
        edge c_fetch_ChargingStation goto fetch;
      location fetch:
        edge c_startCharging goto Charging;
      location Charging:
        edge c_stopCharging goto idle;
    end
    supervisor automaton PathScanning:
      location Deactive:
        initial;
```

```
      marked;
      edge AuxiliaryMovements.c_rotate when CameraSensor.on;
      edge VehicleLever.c_Raise when CameraSensor.on;
      edge VehicleLever.c_Lower when CameraSensor.on;
  end
  supervisor automaton ChargeWhenBatteryIsLow:
    location Getting_Location:
      edge ChargingSystem.c_startCharging when BatteryLevel.low and
          VehicleMovement.idle and AuxiliaryMovements.NoRotation and
          ObstacleSensor.off and TouchSensor.off and CameraSensor.on goto Charging;
    location NoCharging:
      initial;
      marked;
      edge ChargingSystem.c_fetch_ChargingStation when BatteryLevel.low goto
          Getting_Location;
    location Charging:
      edge ChargingSystem.c_stopCharging when BatteryLevel.full goto NoCharging;
  end
  supervisor automaton stopAllMovementsWhenCharging_Lever:
    location Deactive:
      initial;
      marked;
      edge VehicleMovement.c_start when ObstacleSensor.off and TouchSensor.off and
          BatteryLevel.full and ChargingSystem.idle and VehicleStatus.Busy or
          ObstacleSensor.off and TouchSensor.off and BatteryLevel.low and
          ChargingSystem.fetch and ChargeWhenBatteryIsLow.Getting_Location;
      edge VehicleLever.c_Raise when ChargingSystem.idle and
          AuxiliaryMovements.NoRotation and VehicleMovement.idle and
          VehicleStatus.Busy;
      edge VehicleLever.c_Lower when ChargingSystem.idle and
          AuxiliaryMovements.NoRotation and VehicleMovement.idle and
          VehicleStatus.Busy;
      edge AuxiliaryMovements.c_rotate when BatteryLevel.full and VehicleStatus.Busy
          and VehicleLever.intermediate_LowToHigh = false and ChargingSystem.idle
          and VehicleLever.intermediate_HighToLow = false or ChargingSystem.fetch
          and ChargeWhenBatteryIsLow.Getting_Location;
      edge VehicleMovement.c_stop when ObstacleSensor.on or TouchSensor.on or
          CameraSensor.on or BatteryLevel.critical;
  end
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleLever.c_Raise;
  requirement invariant BatteryLevel.critical = true disables
      AuxiliaryMovements.c_rotate;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_isAvailable;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_inProcess;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_ExecutePathPlanner;
  requirement invariant BatteryLevel.low = true or BatteryLevel.critical = true
      disables VehicleStatus.c_GetDestination;
  requirement invariant AuxiliaryMovements.Rotation or BatteryLevel.low = true or
      CameraSensor.off or VehicleLever.intermediate_LowToHigh or
      VehicleLever.intermediate_HighToLow or BatteryLevel.critical = true or
      ChargingSystem.Charging disables VehicleStatus.c_JobCompleted;
  requirement invariant VehicleMovement.move disables AuxiliaryMovements.c_rotate;
  requirement invariant AuxiliaryMovements.Rotation disables VehicleMovement.c_start;
  requirement invariant VehicleLever.intermediate_LowToHigh disables
      VehicleMovement.c_start;
  requirement invariant VehicleLever.intermediate_HighToLow disables
      VehicleMovement.c_start;
```

```
  end
end
supervisor automaton sup:
  alphabet warehouse1.robot1.VehicleMovement.c_start,
      warehouse1.robot1.VehicleMovement.c_stop,
      warehouse1.robot1.AuxiliaryMovements.c_rotate,
      warehouse1.robot1.AuxiliaryMovements.c_stopc_rotate,
      warehouse1.robot1.VehicleLever.c_Raise, warehouse1.robot1.VehicleLever.c_Lower,
      warehouse1.robot1.VehicleStatus.c_isAvailable,
      warehouse1.robot1.VehicleStatus.c_GetDestination,
      warehouse1.robot1.VehicleStatus.c_ExecutePathPlanner,
      warehouse1.robot1.VehicleStatus.c_inProcess,
      warehouse1.robot1.VehicleStatus.c_JobCompleted,
      warehouse1.robot1.CameraSensor.c_Scanning,
      warehouse1.robot1.ChargingSystem.c_fetch_ChargingStation,
      warehouse1.robot1.ChargingSystem.c_startCharging,
      warehouse1.robot1.ChargingSystem.c_stopCharging,
      warehouse1.robot2.VehicleMovement.c_start,
      warehouse1.robot2.VehicleMovement.c_stop,
      warehouse1.robot2.AuxiliaryMovements.c_rotate,
      warehouse1.robot2.AuxiliaryMovements.c_stopc_rotate,
      warehouse1.robot2.VehicleLever.c_Raise, warehouse1.robot2.VehicleLever.c_Lower,
      warehouse1.robot2.VehicleStatus.c_isAvailable,
      warehouse1.robot2.VehicleStatus.c_GetDestination,
      warehouse1.robot2.VehicleStatus.c_ExecutePathPlanner,
      warehouse1.robot2.VehicleStatus.c_inProcess,
      warehouse1.robot2.VehicleStatus.c_JobCompleted,
      warehouse1.robot2.CameraSensor.c_Scanning,
      warehouse1.robot2.ChargingSystem.c_fetch_ChargingStation,
      warehouse1.robot2.ChargingSystem.c_startCharging,
      warehouse1.robot2.ChargingSystem.c_stopCharging;
  location:
    initial;
    marked;
    edge warehouse1.robot1.AuxiliaryMovements.c_rotate when true;
    edge warehouse1.robot1.AuxiliaryMovements.c_stopc_rotate when true;
    edge warehouse1.robot1.CameraSensor.c_Scanning when true;
    edge warehouse1.robot1.ChargingSystem.c_fetch_ChargingStation when true;
    edge warehouse1.robot1.ChargingSystem.c_startCharging when true;
    edge warehouse1.robot1.ChargingSystem.c_stopCharging when true;
    edge warehouse1.robot1.VehicleLever.c_Lower when true;
    edge warehouse1.robot1.VehicleLever.c_Raise when true;
    edge warehouse1.robot1.VehicleMovement.c_start when true;
    edge warehouse1.robot1.VehicleMovement.c_stop when true;
    edge warehouse1.robot1.VehicleStatus.c_ExecutePathPlanner when true;
    edge warehouse1.robot1.VehicleStatus.c_GetDestination when true;
    edge warehouse1.robot1.VehicleStatus.c_inProcess when true;
    edge warehouse1.robot1.VehicleStatus.c_isAvailable when true;
    edge warehouse1.robot1.VehicleStatus.c_JobCompleted when true;
    edge warehouse1.robot2.AuxiliaryMovements.c_rotate when true;
    edge warehouse1.robot2.AuxiliaryMovements.c_stopc_rotate when true;
    edge warehouse1.robot2.CameraSensor.c_Scanning when true;
    edge warehouse1.robot2.ChargingSystem.c_fetch_ChargingStation when true;
    edge warehouse1.robot2.ChargingSystem.c_startCharging when true;
    edge warehouse1.robot2.ChargingSystem.c_stopCharging when true;
    edge warehouse1.robot2.VehicleLever.c_Lower when true;
    edge warehouse1.robot2.VehicleLever.c_Raise when true;
    edge warehouse1.robot2.VehicleMovement.c_start when true;
    edge warehouse1.robot2.VehicleMovement.c_stop when true;
    edge warehouse1.robot2.VehicleStatus.c_ExecutePathPlanner when true;
    edge warehouse1.robot2.VehicleStatus.c_GetDestination when true;
```

```
    edge warehouse1.robot2.VehicleStatus.c_inProcess when true;
    edge warehouse1.robot2.VehicleStatus.c_isAvailable when true;
    edge warehouse1.robot2.VehicleStatus.c_JobCompleted when true;
end
```

References

[1] Raffaello D'Andrea and Peter Wurman. Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities. In *2008 IEEE International Conference on Technologies for Practical Robot Applications*, pages 80–83. IEEE, 2008.