Politecnico di Milano

Academic Year 2017-2018

Formal Methods for Concurrent and Real Time Systems

Collaborative Robotics Modeling

Riva Daniele - matricola 875154

Tavecchia Giorgio - matricola 874716

Introduction and scenario presentation:

Introduction:

The purpose of this document is to model, through “TRIO Temporal Logic” a collaborative robotics scenario; in particular, the objective will be to model a specific scenario, described below, in order to be able to verify the safety requirements.

HRC Scenario:

The scenario here discussed concerns the use of a “KUKA Mobile Robot” able to move freely in the work area and equipped with a manipulator arm to perform its activities.

Despite the high degree of autonomy of the robot, in the same area there is an operator with the task of monitoring robot’s activity and interacting, through an HDI (Human Device Interface) or manually on the robotic arm, if necessary.

The work that must be performed by the robot consists in transporting a series of objects (Working Pieces [WP]) from a collection area (Bin) to the workstation.

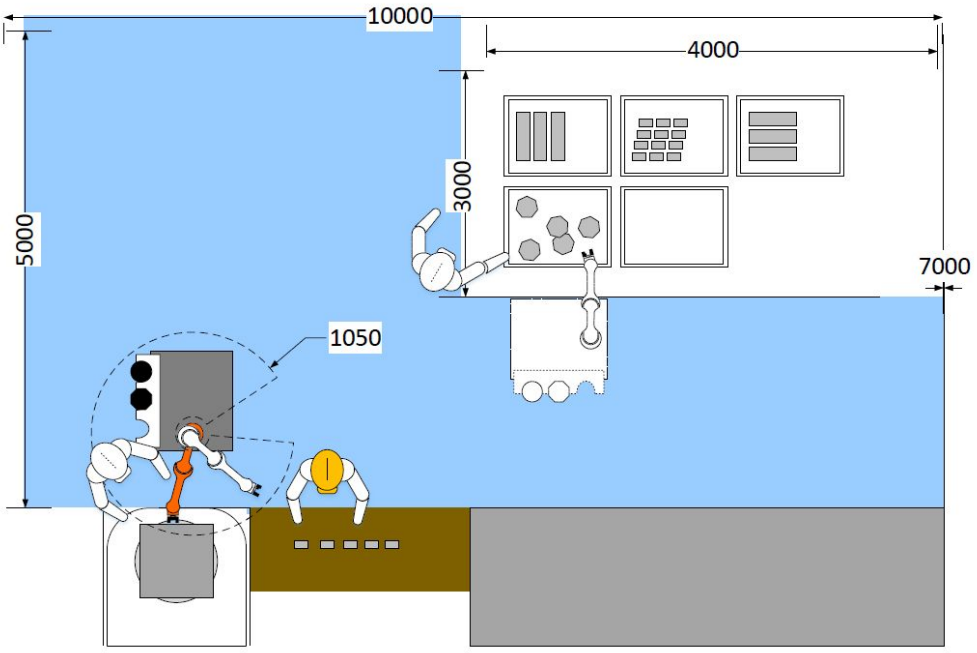

Figure 1: Top-view of the workspace

In the above image, the area where operator and robot can move freely is indicated in blue.

The white area in the upper right corner is the Room Bin where the working pieces are kept. Work station is shown on left down corner, with the conveyor belt on its side.

The gray area is not reachable. In white robot and operator positions are shown.

KUKA Mobile Manipulator details:

Figure 2: The photo on the right shows the structure of a Kuka manipulator arm. The photo on the left shows the same device mounted over a mobile cart which has a local bin of its own.

The KUKA manipulator consists of a base equipped with 4 omnidirectional wheels that allow it to move independently in any direction.

Above this base we find a number of slots, used for the deposit of collected working pieces, and, to the side, the manipulator arm, composed in turn by a series of links that allow the actuator to reach any point of the three-dimensional space around the arm itself.

The actuator has only the functionality to grasp and lay down working pieces.

The speed of the robot is variable and is determined by the specific working condition and by the measurements coming from the sensors placed on the manipulator itself: for example, the speed will be reduced, or in the worst case

stopped, if the robot detects that the operator is at a distance considered risky for his safety. Robot and manipulator speed are independent.
As mentioned the robot is equipped with numerous sensors that allows it to:

- Detect if a working pieces has been correctly grasped
- Know the available/occupied slot numbers on its base
- Detect its speed
- Detect the speed of the manipulator arm
- Detect its position within the work area
- Detect the presence of obstacles (operator, walls, etc…) around itself
- Identify operator's body parts and their distance
- Detect the position of the manipulator arm
- Detect if the manipulator arm and the operator's hand are in contact (i.e. the operator is taking case of addressing the arm itself)

Operator details:

Within the described scenario, there is a single operator with the task of monitoring the robot's activity and, where necessary, correcting it.
He is equipped with an HDI with which it is possible to give commands to the KUKA manipulator, even remotely, in order to correct its behaviour. Thanks to it the operator will be able to:

- Enable it
- Turn it off
- Direct to a specific location in the workspace

or to issue commands to manage:

- Arm movement
- Actuator status
- Working pieces on the base (empty/fill)

The interaction can also be done manually and so the operator will be able to direct the manipulator arm as well as move the robot, if disabled.

The operator does not wear any particular protective measures and for this reason at each part of the body will be assigned a specific level of criticality, greater for head and hands.

The operator is able to move freely within the entire work area.

KUKA Mobile Manipulator tasks

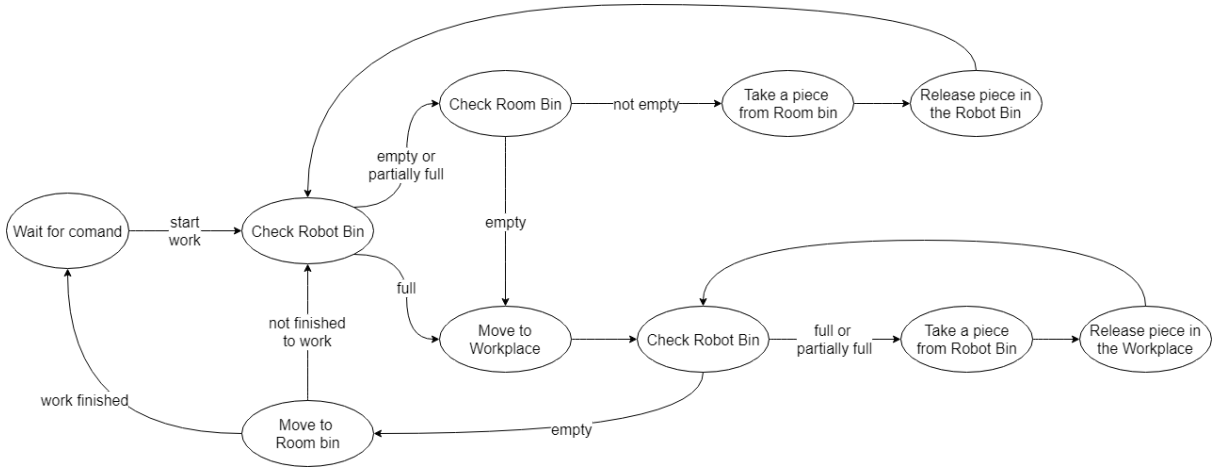

Figure 3: activity diagram of robot task

The robot waits for the work to be performed in the room bin; as soon as it receives the command to start its work, it begins to take pieces using the manipulator arm from the room bin and moves them to its base until the latter has been filled or there are no other pieces to load.

Once the loading phase is over, it begins to move towards the workplace; during the motion it must pay attention to the surrounding area, in this way, taking advantage of different sensors, in addition to following the calculated path, it must avoid or stop before the obstacles it could encounter, both to avoid damaging itself and third parties (other objects, robots or operators).

The speed of the motion will be adjusted according to the area in which it is located (near the walls more slowly, towards the center more quickly). The arm, during movements in the work area, must remain still for the same reasons mentioned above.

When it arrives in the workplace area, it can start the unloading phase and empty the cart completely. After that it will be able to position itself in front of the room bin or continue the job if it has not finished yet.

Throughout this process, its sensors will continue to provide a global reading of the environment so that in situations of proximity to the operator it should slow down is speed or stop.

At each state, if robot receives stop command from operator or from sensors, it stops to do anything (work or move) and waits until receives a command from operator or sensor permission to back to work. Sensors and commands can interact at each state and modify the speed of work or of movement.

The operator through the HDI can completely manage the work of the robot, so it can stop it (moving or being in loading/unloading phase) or even tell it to move from the area if the operator needs the space in which the robot is located.

Modelling

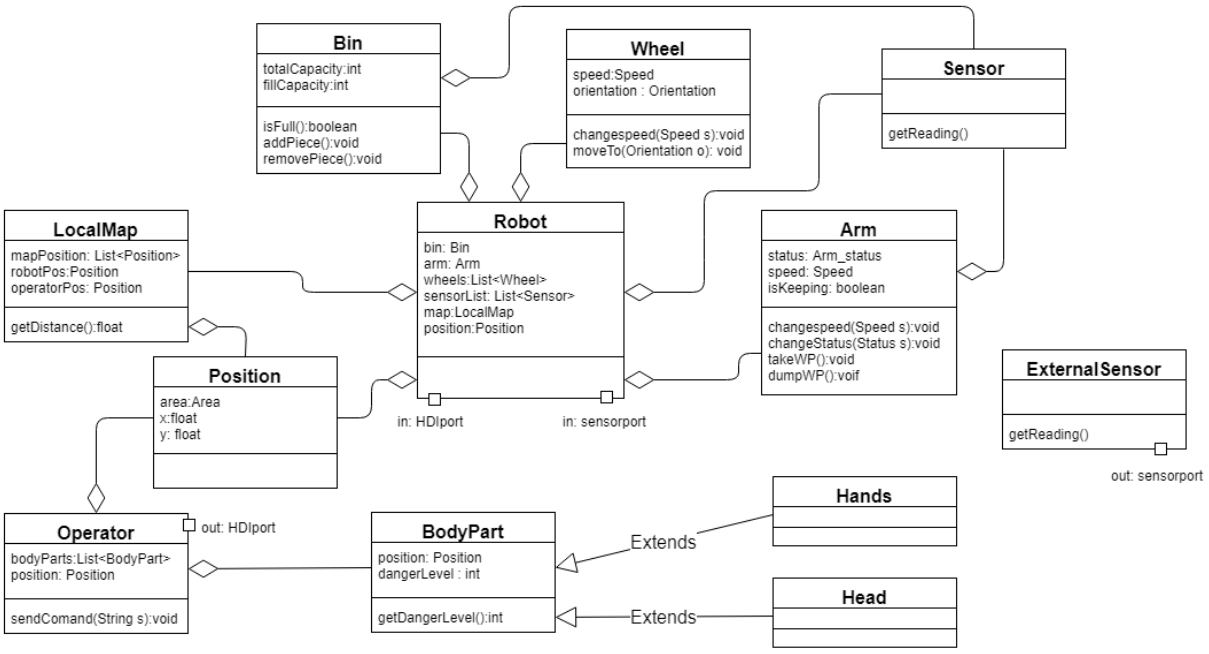

Figure 4: UML

The UML describes how can be implemented robot and environment at software level; it has some more information that are not modeled in TRIO or are handled in

different way for practice reason. For example, TRIO model doesn't take account of precise position
(coordinates in the room) but it's only interested to the area where operator and robot are located and distance between them; another example is the wheels orientation that is helpful for path planning and not for check requirements.
In the UML, two kind of sensors are specified: sensors are those that belong to robot and gives information about status of wheels, arm, bin … External sensors are those spreaded in the room and with the port (we suppose that ports can be a communication transceiver) communicate and give information about environment to robot.
With this information, robot can build and keep updated a map of positions and it can know the distance between itself and operator,walls,workstation…
Finally, operator has a port that represents a HDI used to give commands to robot.

Assumptions:

In order to model the scenario, we now introduce some assumptions related to sensors and robot behaviours.
We assume that all sensor measurements are accurate and fast in their diffusion, this means that a situation in which an event is late reported can't happen, and so the robot is always able to react promptly.
In addition, we suppose that the movements of the manipulator are always correct, that is, the situation in which a working piece is not correctly positioned on the bin or grasped can not occur.

Map overview:

With regard to the work area and the possible reciprocal positions that robots and operators can assume within it, we decided to divide it into a series of sub-areas, each defined according to the characteristics and behaviors that the robot must maintain.
This means that, for example, in the areas near the room bin and the workstation, the robot will have to maintain a greater "level of attention" regarding movements

and speeds of the same in order to avoid to lose an object grasped or obstruct the operator’s movements, thus avoiding, in the worst case, injuries to the latter.
In particular the work area will be defined as follows:

1. **Room bin**:
    - In this area the robot must pay the maximum attention to its movements and to the speed of the same as the operator could be in proximity to move the working pieces inside the bin or approach suddenly to correct an incorrect action of the robot itself .
    - In this area the robot will be stationary for most of the time while, on the contrary, the manipulator arm will presumably be moving to grasp and load working pieces on its bin.
2. **Workstation**:
    - The characteristics of this area are completely identical to the Room Bin area as the operator could decide to manually address the manipulator arm.
3. **Wall Area**:
    - This area consists of all the positions near a wall, i.e. those positions where the robot, presumably in motion and with the retracted manipulator arm, must pay attention not to collide with the operator or to not close it against the wall, preventing him from moving.
4. **Limited Area**:
    - This area represents the entire perimeter of the work area in contact with workstations, bin areas and wall areas, i.e. positions in which the robot's attention is high but not maximum, as it could approach hazardous areas. This means that the robot will start to slow down or accelerate depending on the area it comes from and where it is going.
5. **Free Area**:
    - This area corresponds to positions in which the robot is far from possible dangers, not considering the operator, and in which therefore the speed of movement will be maximum. The operator will not be in most of the time near the robot and therefore the speed of the latter will be limited only in case this event occurs.

We emphasize that the areas do not overlap and, as will be defined later in the axioms, the shift from one to the other will be possible only if the latter have a common side.

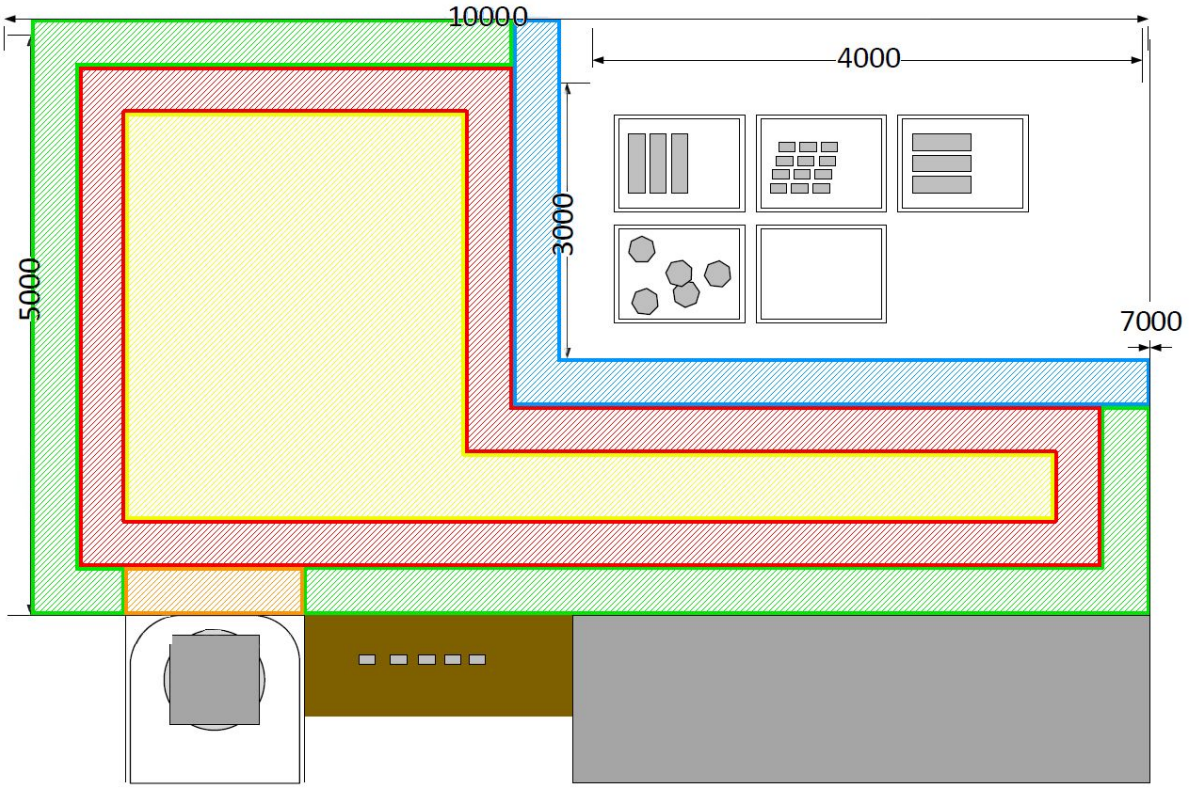

Figure 5:

Blue area : Room bin

Orange area: Workstation

Green area: Wall Area

Red area: Limited Area

Yellow Area: Free Area

Modelling states:

To model the scenario we now introduce a series of states that will allow us to define the mutual positions of robot and operator in terms of positions, as well as the distance between them detected by the robot sensors.

States will not only concern positions and distances but they will also consider the bin positioned on the robot, in order to monitor its state, the manipulator arm and the actuator connected to it.

This will allow us to define specific behavior of the robot as a function of the global state, as result of the interpretation and combination of other local states.
We therefore define the following states:

- **S_Robot_Position** = *{ room_bin, workstation, wall_area, limited_area, free_area}*
- **S_Operator_Position** = *{ room_bin, workstation, wall_room, limited_area, free_area}*
- **S_Distance_Robot_Operator** = *{safe, near, critical, direct_contact}*
- **S_Nearest_Body_Part** = *{head, hands, null}*
    - The *null* value indicates that the sensor is not able to perceive the operator proximity, meaning that he is far away. This state is defined in relation to the actuator position.

States related to the position of the robot and the operator coincide, this will allow us to detect if they are in the same area and **S_Distance_Robot_Operator** will provide us with more information about their distance and, therefore, to adjust the behavior of the robot in every situation.

- **S_Speed_Robot** = *{fast, limited, stop, accelerating,,braking}*
- **S_Speed_Arm** = *{fast, limited, stop, accelerating,,braking}*

The speed control allow us to ensure that the robot is able to detect possible collisions with the operator in time and therefore to stop.

- **S_Arm** = *{outstretched, at_rest, arm_moving }*
    - we consider that the robot is able to put a working piece in its bin when the manipulator is *at_rest* position
- **S_Arm_Actuator** = *{free, kept, actuator_moving}*
    - *kept* status indicates that the actuator is grabbing a working piece.
- **S_Robot_Bin** = *{empty, fill, partially_fill}*
- **S_Room_Bin** = *{empty, fill}*

The different states of the robot and room bin allow us to indicate two different situations: the room bin is considered *fill* when it contains at least one working piece while the robot bin is considered *fill* when it can not add other working pieces, in opposite it will be *partially_fill* if at least one is transported, *empty* otherwise.
We consider that the robot arm is stationary and retracted when the robot is moving and therefore can only be extended when it is stopped.

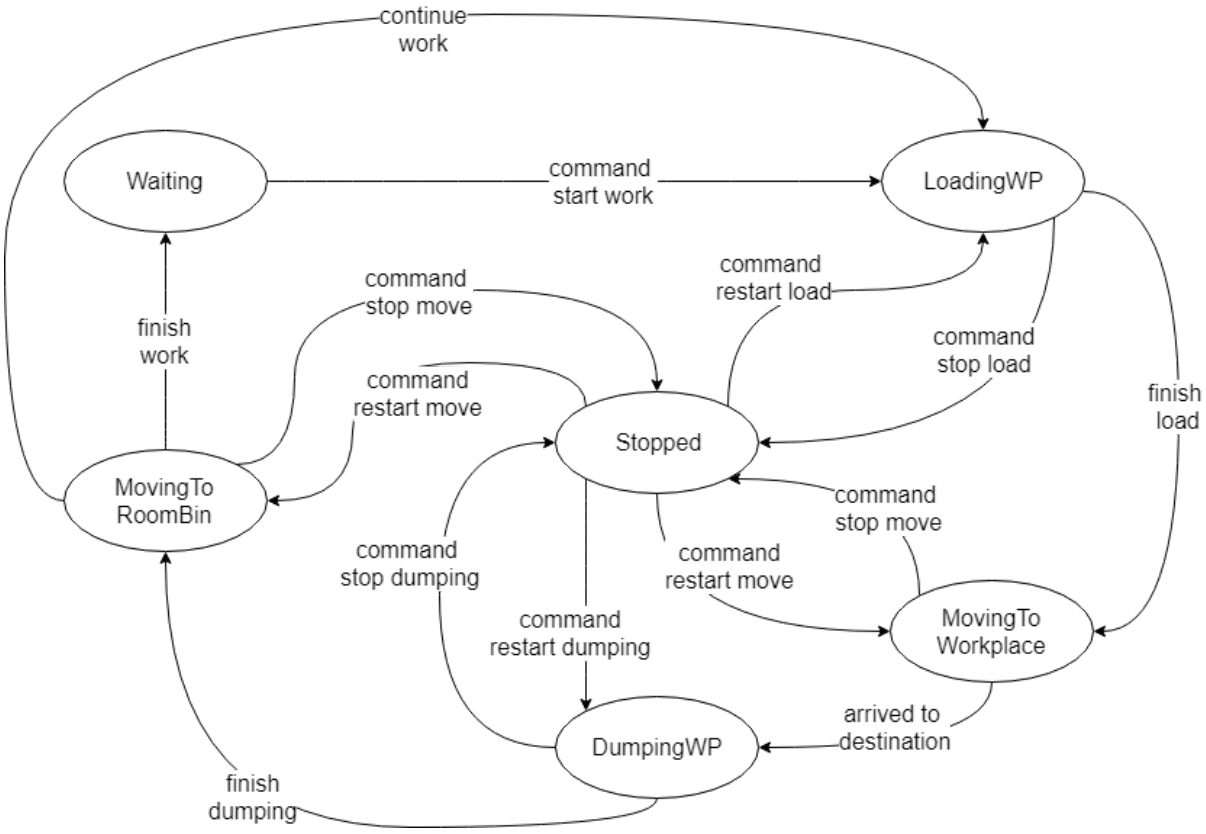

Figure 6: robot state chart

- **S_Robot_Status**=*{loading,dumping,stopped, movingToRoomBin, movingToWorkstation, waiting}*
    - *waiting:* the robot is in room bin area, waiting to start its task. The robot was just turned on or the room bin is now empty.

Modelling events:

In order to produce the states and the changes between them within our model, we introduce here the events that will then be used within the modeling in TRIO.

The names used mainly concern actions and are therefore self-explanatory in most cases.

- **E_Robot_Enter_RoomBin**
- **E_Robot_Enter_WorkStation**
- **E_Robot_Near_Wal_Room**
- **E_Robot_Enter_LimitedArea**
- **E_Robot_Enter_FreeArea**
- **E_Operator_Enter_RoomBin**
- **E_Operator_Enter_WorkStation**
- **E_Operator_Near_Wal_Room**
- **E_Operator_Enter_LimitedArea**
- **E_Operator_Enter_FreeArea**

These events are the result of readings from the sensors placed on the robot:

- **E_Sensor_Distance_Safe**
- **E_Sensor_Distance_Near**
- **E_Sensor_Distance_Critical**
- **E_Sensor_Distance_Direct_Contact**
- **E_Sensor_Room_Bin_Full**
- **E_Sensor_Room_Bin_Empty**
- **E_Sensor_Near_Head**
- **E_Sensor_Near_Hands**

Modelling commands:

As previously specified, the operator is able to modify the working stub of the robot by imparting certain commands via the HDI or manually by acting on the robot arm. We now list the possible commands that can be used:

- **command** (*{stop, work})*
    - *stop:* the command forces the robot to stop, regardless of the operation performed and to maintain the current state.
    - *work:* the command forces the robot to resume its autonomous operation.
- **go** (*{ room_bin, workstation,}*)
    - *room_bin*: instructs the robot to go to room bin in order to fill its bin and continue the autonomous work
    - *workstation*: like the previous one but going to the workstation
- **move_arm**({*stretch, retire})*
- **action_arm(***{release_WP, take_WP})*
- **change_speed***({fast, limited, stop})*
    - It can be applied to the robot itself or to the manipulator arm.
- **bin_command**(*{add, remove}*)
    - Modify logical information about the robot bin.

TRIO modeling:

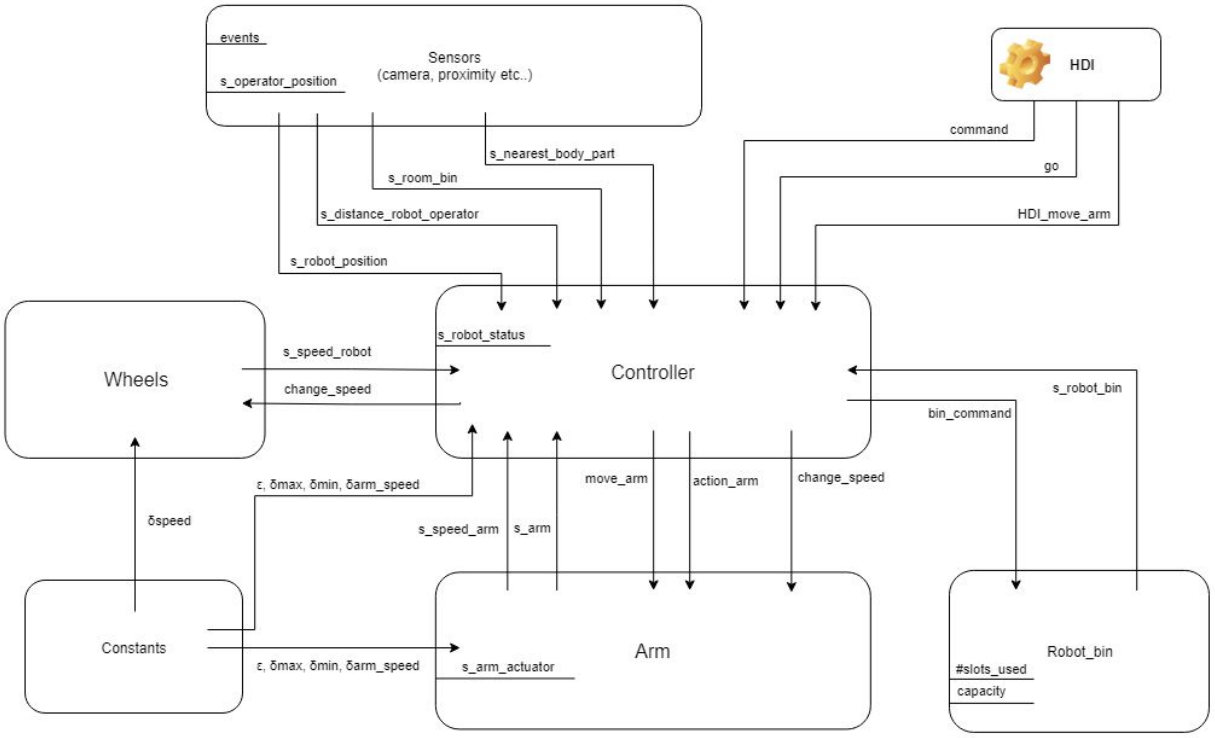

Figure 7: Graphical TRIO

The graphic version of the TRIO classes is shown in the figure, which we will now illustrate in detail.

The robot consists of a series of components that exchange states and commands through a central controller.

This, receiving a whole series of local states coming from wheels, bin, manipulator arm and information coming from the surrounding environment, obtained through the use of different sensors, will have the task to provide the appropriate instructions for the achievement of the various activities of the robot, without block or causing damage to the operator.

We define the following constants that will be later used:

δ_{max}*: arm's time to change from at_rest to outstretched if speed is limited*

δ_{min}*: arm's time to change from at_rest to outstretched if speed is fast*

δ_{arm_speed}*: robot's time to change arm's speed*

ε *: arm_actuator's time to take or release wp*

δ_{speed} *: robot's time to change its speed move*

Class Constants

visible $\varepsilon, \delta_{max}, \delta_{min}, \delta_{arm\_speed}, \delta_{speed}$

temporal domain real;

TI items

consts $\varepsilon, \delta_{max}, \delta_{min}, \delta_{arm\_speed}, \delta_{speed}$ *: real*

axioms:

$\varepsilon > 0$

$\delta_{max} > 0$

$\delta_{min} > 0$

$\delta_{arm\_speed} > 0$

$\delta_{max} > \delta_{min}$

$\delta_{speed} > 0$

end Constants

Class Arm

visible move_arm, change_speed, s_arm, s_speed_arm

temporal domain real;

TD items

predicate move_arm({*stretch, retire})*

action_arm({release_WP, take_WP})

change_speed*({fast, limited, stop})*

vars s_arm = *{outstretched, at_rest, arm_moving }*

s_arm_actuator = *{free, kept, actuator_moving}*

s_speed_arm = *{fast, limited, stop, accelerating,,braking}*

TI items

consts $\varepsilon, \delta_{max}, \delta_{min}, \delta_{arm\_speed}$ *: real*

axioms:

- *The following axioms ensure the correct movement of the manipulator arm upon receipt of a specific command. δ_{max} and δ_{min} , as previously specified, are used to indicate the different time intervals necessary for the elongation and withdrawal of the manipulator arm*

$$\text{UpToNow(at\_rest)} \wedge \text{move\_arm(stretch)} \wedge \text{s\_speed\_arm=limited} \wedge \neg \text{Within}_{ee}(\text{move\_arm(retire)}, \delta_{max}) \rightarrow \text{Lasts}_{ie}(\text{arm\_moving}, \delta_{max}) \wedge \text{Futr}(\text{Until}(\text{outstreched}, \text{move\_arm(retire)}), \delta_{max})$$

$$\text{UpToNow(at\_rest)} \wedge \text{move\_arm(stretch)} \wedge \text{s\_speed\_arm=fast} \wedge \neg \text{Within}_{ee}(\text{move\_arm(retire)}, \delta_{min}) \rightarrow \text{Lasts}_{ie}(\text{arm\_moving}, \delta_{min}) \wedge \text{Futr}(\text{Until}(\text{outstreched}, \text{move\_arm(retire)}), \delta_{min})$$

$$\text{UpToNow(outstretched)} \wedge \text{move\_arm(retire)} \wedge \text{s\_speed\_arm=limited} \wedge \neg \text{Within}_{ee}(\text{move\_arm(strech)}, \delta_{max}) \rightarrow \text{Lasts}_{ie}(\text{arm\_moving}, \delta_{max}) \wedge \text{Futr}(\text{Until}(\text{at\_rest}, \text{move\_arm(stretch)}), \delta_{max})$$

$$\text{UpToNow(outstretched)} \wedge \text{move\_arm(retire)} \wedge \text{s\_speed\_arm=fast} \wedge \neg \text{Within}_{ee}(\text{move\_arm(strech)}, \delta_{min}) \rightarrow \text{Lasts}_{ie}(\text{arm\_moving}, \delta_{min}) \wedge \text{Futr}(\text{Until}(\text{at\_rest}, \text{move\_arm(stretch)}), \delta_{min})$$

- *We introduce a small delay indicated with ε to model the operation of the actuator and its grasping phases*

$$\text{UpToNow(kept)} \wedge \text{action\_arm(release\_WP)} \rightarrow \text{Lasts}_{ie}(\text{actuator\_moving}, \varepsilon) \wedge \text{Futr}(\text{Until}(\text{free}, \text{move\_arm(take\_WP)}), \varepsilon)$$

$$\text{UpToNow(free)} \wedge \text{action\_arm(take\_WP)} \rightarrow \text{Lasts}_{ie}(\text{actuator\_moving}, \varepsilon) \wedge \text{Futr}(\text{Until}(\text{kept}, \text{action\_arm(release\_WP)}), \varepsilon)$$

- *The following axioms specify how the speed variation is not instantaneous:*

(UpToNow(fast) ∨ UpToNow(limited)) ∧ change_speed(stop) →

Lasts$_{ie}$(braking,δ_{arm_speed}) ∧

(

Futr(Until(stop, change_speed(limited)),δ_{arm_speed})

∨ Futr(Until(stop, change_speed(fast)),δ_{arm_speed})

)

(UpToNow(stop) ∨ UpToNow(limited)) ∧ change_speed(fast) →

Lasts$_{ie}$(accelerating,δ_{arm_speed}) ∧

(

Futr(Until(fast, change_speed(limited)),δ_{arm_speed})

∨ Futr(Until(fast, change_speed(stop)),δ_{arm_speed})

)

UpToNow(fast) ∧ change_speed(limited) →

Lasts$_{ie}$(braking,δ_{arm_speed}) ∧

(

Futr(Until(limited, change_speed(stop)),δ_{arm_speed})

∨ Futr(Until(limited, change_speed(fast)),δ_{arm_speed})

)

UpToNow(stop) ∧ change_speed(limited) →

Lasts$_{ie}$(accelerating,δ_{arm_speed}) ∧

(

Futr(Until(limited, change_speed(stop)),δ_{arm_speed})

∨ Futr(Until(limited, change_speed(fast)),δ_{arm_speed})

)

end Arm

Class Robot_Bin

visible bin_command, s_robot_bin

temporal domain real;

TD items

predicate bin_command(*{add, remove}*)

vars s_robot_bin = *{empty, fill, partially_fill}*

TI items

vars #slots_used : natural

consts capacity : natural

axioms:

- *The bin is represented here by a predicate that allows instant updating of the information relative to the capacity used. It will be controller's task, as specified below, to ensure that this command is only called upon the effective addition or removal of a working piece within the robot bin:*

Alw(#slots_used ≤ capacity)

UpToNow(empty) ∧ bin_command(add) → (#slots_used++ ∧ partially_fill)

UpToNow(partially_fill) ∧ bin_command(add) →

(#slots_used++ ∧ ((#slots_used == capacity) → fill))

UpToNow(partially_fill) ∧ bin_command(remove) →

(#slots_used-- ∧((#slots_used == 0) → empty))

UpToNow(fill) ∧ bin_command(remove) → (#slots_used-- ∧ partially_fill)

end Robot_Bin

Class Wheels

visible s_speed_robot, change_speed

temporal domain real;

TD items

predicate change_speed (*{fast, limited, stop})*

vars s_speed_robot (*{fast, limited, stop, accelerating,,braking})*

TI items

consts δ_{speed}*: real*

axioms:

- *The following axioms, completely identical to those reported in the class relative to the manipulator arm, are used to indicate the transaction between one speed and the other following reception of the relative command and of how this is not instantaneous:*

$$(\text{UpToNow(fast)} \vee \text{UpToNow(limited)}) \wedge \text{change\_speed(stop)} \rightarrow$$
$$\text{Lasts}_{ie}(\text{braking}, \delta_{speed}) \wedge$$
$$($$
$$\text{Futr(Until(stop, change\_speed(limited))}, \delta_{speed})$$
$$\vee \text{Futr(Until(stop, change\_speed(fast))}, \delta_{speed})$$
$$)$$

$$(\text{UpToNow(stop)} \vee \text{UpToNow(limited)}) \wedge \text{change\_speed(fast)} \rightarrow$$
$$\text{Lasts}_{ie}(\text{accelerating}, \delta_{speed}) \wedge$$
$$($$
$$\text{Futr(Until(fast, change\_speed(limited))}, \delta_{speed})$$
$$\vee \text{Futr(Until(fast, change\_speed(stop))}, \delta_{speed})$$
$$)$$

UpToNow(fast) ∧ change_speed(limited) →

Lasts$_{ie}$(braking,δ_{speed}) ∧

(

Futr(Until(limited, change_speed(stop)),δ_{speed})

∨ Futr(Until(limited, change_speed(fast)),δ_{speed})

)

UpToNow(stop) ∧ change_speed(limited) →

Lasts$_{ie}$(accelerating,δ_{speed}) ∧

(

Futr(Until(limited, change_speed(stop)),δ_{speed})

∨ Futr(Until(limited, change_speed(fast)),δ_{speed})

)

end Wheels

Class Sensors

visible s_robot_position, s_distance_robot_operator,
s_room_bin, s_nearest_body_part

temporal domain real;

TD items

predicate E_Robot_Enter_RoomBin, E_Robot_Enter_WorkStation,
E_Robot_Near_Wall_Room, E_Robot_Enter_LimitedArea,
E_Robot_Enter_FreeArea, E_Operator_Enter_RoomBin,
E_Operator_Enter_WorkStation, E_Operator_Near_Wall_Room,
E_Operator_Enter_LimitedArea, E_Operator_Enter_FreeArea,
E_Sensor_Distance_Safe, E_Sensor_Distance_Near,
E_Sensor_Distance_Critical, E_Sensor_Distance_Direct_Contact,
E_Sensor_Near_Head, E_Sensor_Near_Hands,
E_Sensor_Room_Bin_Full, E_Sensor_Room_Bin_Empty

vars s_robot_position = *{room_bin, workstation,*
wall_area, limited_area, free_area},
s_operator_position = *{room_bin, workstation,*
wall_area, limited_area, free_area},
s_distance_robot_operator = {safe, near, critical, direct_contact},
s_nearest_body_part = *{head, hands,null},*
s_room_bin = *{empty, fill},*

axioms:

- *The following axioms specify how the status update after the related events is instantaneous:*

Dist(e_robot_enter_roombin, t) ↔ Dist(s_robot_position = room_bin, t)

Dist(e_robot_enter_workstation, t) ↔ Dist(s_robot_position = workstation, t)

Dist(e_robot_near_wall_room, t) ↔ Dist(s_robot_position = wall_area, t)

Dist(e_robot_enter_limitedArea, t) ↔ Dist(s_robot_position = limited_area, t)

Dist(e_robot_enter_freeArea, t) ↔ Dist(s_robot_position = free_area, t)

Dist(e_operator_enter_roombin, t) ↔ Dist(s_operator_position = room_bin, t)
Dist(e_operator_enter_workstation, t) ↔ Dist(s_operator_position = workstation, t)
Dist(e_operator_near_wall_room, t) ↔ Dist(s_operator_position = wall_area, t)
Dist(e_operator_enter_limitedArea, t) ↔ Dist(s_operator_position = limited_area, t)
Dist(e_operator_enter_freeArea, t) ↔ Dist(s_operator_position = free_area, t)

Dist(e_sensor_distance_safe, t) ↔ Dist(s_distance_robot_operator = safe, t)
Dist(e_sensor_distance_near, t) ↔ Dist(s_distance_robot_operator = near, t)
Dist(e_sensor_distance_critical, t) ↔ Dist(s_distance_robot_operator = critical, t)
Dist(e_sensor_distance_direct_contact, t) ↔
Dist(s_distance_robot_operator = direct_contact, t)

Dist(e_sensor_near_head, t) ↔ Dist(s_nearest_body_part = head, t)
Dist(e_sensor_near_hands, t) ↔ Dist(s_nearest_body_part = hands, t)
Dist(e_sensor_distance_safe, t) ↔ Dist(s_nearest_body_part = null, t)

Dist(e_sensor_room_bin_full, t) ↔ Dist(s_room_bin = full, t)
Dist(e_sensor_room_bin_empty, t) ↔ Dist(s_room_bin = empty, t)

- *With these axioms we guarantee that the sensor readings take place correctly, that is, at each time instant there can only be one event for each type*

Alw(e_robot_enter_roombin ∨
e_robot_enter_workstation ∨ e_robot_near_wall_room ∨
e_robot_enter_limitedArea ∨ e_robot_enter_freeArea)

Alw(e_operator_enter_roombin∨
e_operator_enter_workstation ∨ e_operator_near_wall_room ∨
e_operator_enter_limitedArea ∨ e_operator_enter_freeArea)

Alw(e_sensor_distance_safe ∨ e_sensor_distance_near ∨

e_sensor_distance_critical ∨ e_sensor_distance_direct_contact)

Alw(e_sensor_near_head ∨ e_sensor_near_hands ∨
e_sensor_near_null)

Alw(e_sensor_room_bin_empty ∨ e_sensor_room_bin_full)

- *The following axioms are intended to ensure that only certain sequences of events, and therefore movements, are allowed within the work area. This means that the movements will be possible only if they concern areas that are contiguous and that will not be instantaneous:*

UpToNow(s_robot_position = room_bin) →
(NextTime(e_robot_enter_limitedArea,t) ∨
NextTime(e_robot_near_wallArea,t))
∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_robot_enter_workstation, t1) ∨
Dist(¬e_robot_enter_freeArea, t1))

UpToNow(s_robot_position=workstation) →
((NextTime(e_robot_enter_limitedArea,t) ∨
NextTime(e_robot_near_wall_room,t))
∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_robot_enter_roombin, t1) ∨
Dist(¬e_robot_enter_freeArea, t1))

UpToNow(s_robot_position = wall_area) →
((NextTime(e_robot_enter_roombin,t) ∨
NextTime(e_robot_enter_limitedArea,t) ∨
NextTime(e_robot_enter_workstation,t))
∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_robot_enter_free_area, t1))

UpToNow(s_robot_position=free_area) →

(NextTime(e_robot_enter_limitedArea,t)

∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_robot_enter_workstation, t1) ∨

Dist(¬e_robot_near_wall_room, t1) ∨

Dist(¬e_robot_enter_roombin, t1))

- *We emphasize how the area defined as "Limited Area" does not have any constraint since it is possible to reach every other area from there.*

UpToNow(s_operator_position = room_bin) →

(NextTime(e_operator_enter_limitedArea,t) ∨

NextTime(e_operator_near_wallArea,t))

∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_operator_enter_workstation, t1) ∨

Dist(¬e_operator_enter_freeArea, t1))

UpToNow(s_operator_position = workstation) →

((NextTime(e_operator_enter_limitedArea,t) ∨

NextTime(e_operator_near_wall_room,t))

∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_operator_enter_roombin, t1) ∨

Dist(¬e_operator_enter_freeArea, t1))

UpToNow(s_operator_position = wall_area) →

((NextTime(e_operator_enter_roombin,t) ∨

NextTime(e_operator_enter_limitedArea,t) ∨

NextTime(e_operator_enter_workstation,t))

∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_operator_enter_free_area, t1))

UpToNow(s_operator_position = free_area) →

(NextTime(e_operator_enter_limitedArea,t)

∧ ∀t1 ((0 < t1 < t) → (Dist(¬e_operator_enter_workstation, t1) ∨

Dist(¬e_operator_near_wall_room, t1) ∨

Dist(¬e_operator_enter_roombin, t1))

- *We consider the following statements and relative axioms:*

In order to be have a critical distance, robot and operator must be in same areas or contiguous one.

In order to have a direct contact, robot and operator must be in the same area

Dist(s_distance_robot_operator = near,t) →
Dist(s_robot_position = room_bin,t) ∧ Dist(s_operator_position = wall_area, t)) ∨
Dist(s_robot_position = room_bin,t) ∧ Dist(s_operator_position = limited_area, t))
∨
Dist(s_robot_position = workstation,t) ∧ Dist(s_operator_position = wall_area, t)) ∨
Dist(s_robot_position = workstation,t) ∧ Dist(s_operator_position = limited_area, t))
Dist(s_robot_position = wall_area,t) ∧ Dist(s_operator_position = room_bin, t)) ∨
Dist(s_robot_position = wall_area,t) ∧ Dist(s_operator_position = work_station, t))
∨
Dist(s_robot_position = wall_area,t) ∧ Dist(s_operator_position = limited_area, t))
∨
Dist(s_robot_position = limited_area,t) ∧ Dist(s_operator_position = room_bin, t))
∨
Dist(s_robot_position = limited_area,t) ∧ Dist(s_operator_position = work_station,
t))
∨ Dist(s_robot_position = limited_area,t) ∧ Dist(s_operator_position = free_area,
t)) ∨ Dist(s_robot_position = limited_area,t) ∧ Dist(s_operator_position =
wall_area, t)) ∨ Dist(s_robot_position = free_area,t) ∧ Dist(s_operator_position =
limited_area, t)))

Dist((s_distance_robot_operator = direct_contact ∨
s_distance_robot_operator = critical) ,t)
→ (Dist(s_robot_position == s_operator_position),t)

end Sensors

Class Controller

- *The controller, as previously pointed out, will play a central role for the correct operation of the manipulator robot, for this reason several axioms are now listed that will cover all possible cases.*

visible s_robot_position, s_distance_robot_operator,
s_room_bin, s_nearest_body_party, command, go, HDI_move_arm,
s_robot_bin, bin_command, change_speed_arm, action_arm,
move_arm, s_arm, s_speed_arm,
change_speed_robot, s_speed_robot
temporal domain real;

TD items
predicate command(*{stop, work}*),
go (*{ room_bin, workstation}),*
HDI_move_arm({*stretch, retire}),*
move_arm({*stretch, retire}),*
action_arm*({release_WP, take_WP}),*
change_speed_arm*({fast, limited, stop}),*
bin_command(*{add, remove}*),
change_speed_robot (*{fast, limited, stop})*
vars s_robot_position = *{room_bin , workstation,*
wall_area, limited_area, free_area},
s_distance_robot_operator = {safe, near, critical, direct_contact},
s_nearest_body_part = *{head, hands,null},*
s_room_bin = *{empty, fill},*
s_arm = *{outstretched, at_rest, arm_moving}*
s_speed_arm = *{fast, limited, stop, accelerating, braking},*
s_speed_robot = *{fast, limited, stop, accelerating, braking},*
s_robot_bin = *{empty, fill, partially_fill},*
s_robot_status = *{loading, dumping, stopped,*

movingToRoomBin, movingToWorkstation, waiting}

TI items

consts $\varepsilon, \delta_{max}, \delta_{min}, \delta_{arm\_speed}$ *: real*

axioms:

- *This first axiom is designed to model the start of the robot activity, it is located in the "RoomBin" area and only after receiving the start command, i.e. "work", it starts loading working pieces on its own bin*

s_robot_status=waiting ∧ command(work) ∧ (s_robot_position=room_bin) → s_robot_status=loading

- *The following axioms shape the robot's operation in the loading area, the variable considered is the speed of the arm itself, influenced by the axioms described below. We underline how the state of the robot bin is updated only after the action has been completed, in order to avoid possible conditions in which the robot is full, and then start to move, before having placed the object grasped.*

s_robot_status=loading ∧ (s_robot_bin=empty ∨ s_robot_bin=partially_fill) ∧ s_room_bin=fill ∧ s_speed_arm=limited →

∃t1,t2[t1 > δ_{max} ∧ t2 > t1+ε+δ_{max} ∧

move_arm(stretch) ∧ Dist(action_arm(keep),t1) ∧

Dist(move_arm(retire),t1+ε) ∧

Dist(action_arm(release),t2) ∧ Dist(bin_coomand(add),t2)

]

s_robot_status=loading ∧ (s_robot_bin=empty ∨ s_robot_bin=partially_fill) ∧ s_room_bin=fill ∧ s_speed_arm=fast →

∃t1,t2[t1 > δ_{min} ∧ t2 > t1+ε+δ_{min} ∧

move_arm(stretch) ∧ Dist(action_arm(keep),t1) ∧

Dist(move_arm(retire),t1+ε) ∧

Dist(action_arm(release),t2) ∧ Dist(bin_coomand(add),t2)

]

- *We now model the conditions to be verified to make the robot move to the workstation, regardless of where it is, because the robot may have just finished loading, as well as having just been reactivated following a stop command .*

s_robot_status = loading ∧ (s_robot_bin = fill ∨ (s_room_bin = empty ∧ s_robot_bin=partially_fill))

→ s_robot_status = movingToWorkstation

- *Since the robot can receive commands at any time, we specify the following validity condition*

s_robot_status = movingToWorkstation ∧ ¬Within(go(room_bin) , t)

→ Futr(s_robot_position=workstation ∧ s_robot_status = dumping,t)

- *Here we find a list of axioms, specular to previous ones, that define the behavior of the robot during the dumping phase in the workstation.*

s_robot_status=dumping ∧ (s_robot_bin=fill ∨ s_robot_bin=partially_fill) ∧ s_speed_arm=limited →

∃t1,t2[t1 > δ_{max}+ε ∧ t2 > t1+ε ∧

action_arm(keep) ∧

Dist(move_arm(stretch),ε) ∧

Dist(action_arm(release),t1) ∧ Dist(move_arm(retire),t2) ∧

Dist(bin_coomand(remove),t2)

]

s_robot_status=dumping ∧ (s_robot_bin=fill ∨ s_robot_bin=partially_fill) ∧ s_speed_arm=fast →

∃t1,t2[t1 > δ_{min}+ε ∧ t2 > t1+ε ∧

action_arm(keep) ∧

Dist(move_arm(stretch),ε) ∧

Dist(action_arm(release),t1) ∧ Dist(move_arm(retire),t2) ∧

Dist(bin_coomand(remove),t2)

]

s_robot_status = dumping ∧ s_robot_bin = empty

→ s_robot_status = movingToRoomBin

s_robot_status = movingToRoomBin ∧ ¬Within(go(workstation) , t)

→ Futr(s_robot_position=room_bin ∧ s_robot_status = loading,t)

- *The following axiom models the return to the initial waiting phase in which the robot has no working pieces to deliver and the room bin is empty*

s_robot_status = loading ∧ s_robot_bin = empty ∧ s_room_bin = empty

→ s_robot_status = waiting

- *The next axioms are used to ensure that the position of the robot does not change during loading and dumping operations, until the robot status is updated*

s_robot_status = loading →

Until(s_robot_position = room_bin, s_robot_status = movingToWorkstation)

s_robot_status = dumping →

Until(s_robot_position = workstation, s_robot_status =movingToRoomBin),t)

- *The same applies to the "Stopped" and "Waiting" statuses*

(s_robot_status = stopped ∨ s_robot_status = waiting) ∧
s_robot_position=room_bin

→(Until(s_robot_position = room_bin,(go(workstation) ∨ command(work)))

(s_robot_status = stopped ∨ s_robot_status = waiting) ∧
s_robot_position=workstation

→(Until(s_robot_position = workstation,(go(room_bin) ∨
command(work)))

(s_robot_status = stopped ∨ s_robot_status = waiting) ∧
s_robot_position=wall_area

→(Until(s_robot_position = wall_area,(go(workstation) ∨ go(room_bin) ∨
command(work)))

(s_robot_status = stopped ∨ s_robot_status = waiting) ∧

s_robot_position=limited_area

→(Until(s_robot_position = limited_area,(go(workstation) ∨
go(room_bin) ∨ command(work)))

(s_robot_status = stopped ∨ s_robot_status = waiting) ∧
s_robot_position=free_area
→(Until(s_robot_position = free_area,(go(workstation) ∨ go(room_bin) ∨
command(work)))

- *This simple axiom ensures that the manipulator arm is always at rest during movements from one work area to another*

(s_robot_status = movingToRoomBin ∨ s_robot_status = movingToWorkstation)
→ s_arm=at_rest

- *As mentioned previously, the robot can receive commands at any time and as such they provide, where possible, the immediate change of state by the robot to perform the required action.*

command(stop) →
Until(s_robot_status=stopped,(command(work) ∨ go(room_bin) ∨
go(workstation))

command(work) ∧ s_robot_status = stopped ∧ s_robot_bin = empty
→ s_robot_status = movingToRoomBin

command(work) ∧ s_robot_status = stopped ∧ (s_robot_bin = fill ∨ (s_room_bin =
empty ∧ s_robot_bin=partially_fill))
→ s_robot_status = movingToWorkstation

go(room_bin) ∧ (s_robot_status=stopped ∨ s_robot_status=waiting)
→ s_robot_status = movingToRoombin

go(workstation) ∧ (s_robot_status=stopped ∨ s_robot_status=waiting)

→ s_robot_status = movingToWorkStation

- *A necessary condition for the operator to order the robot to stretch or withdraw the arm is that the robot is still, this to avoid possible accidents*

HDI_move_arm(stretch) ∧ (s_robot_status=stopped ∨ s_robot_status=waiting)
→ move_arm(stretch)

HDI_move_arm(retire) ∧ (s_robot_status = stopped ∨ s_robot_status = waiting)
→ move_arm(retire)

- *<u>Speed limitation and safety requirements:</u> finally, we present a series of axioms that guarantee safer work between the robot and the operator. The speed of the robot will be constantly monitored and reduced in specific areas with possible accidents, as well as in the vicinity of the operator. When the distance between the two should be dangerously reduced, the robot will have to stop to avoid possible injuries to the operator. Particular priority will be given to the proximity to the operator's head.*
  We would like to clarify how the speed of movement of the operator was not taken into consideration, but rather its distance from the robot. This is because the modeled behavior is highly proactive, i.e. the speed of the robot is reduced even before the operator can find himself in dangerous situations, thus allowing, in the latter case, to stop in the immediacy.

((s_robot_position = limited_area ∨ s_robot_position = wall_area)
∧ (s_distance_robot_operator = safe ∨ s_distance_robot_operator = near))
→ change_speed_robot(limited)

(s_robot_position = room_bin ∧ (s_robot_status = movingToWorkstation ∨
s_robot_status = movingToRoombin) ∧ (s_distance_robot_operator = safe ∨
s_distance_robot_operator = near))

→change_speed_robot(limited)

(s_robot_position = workstation ∧ s_robot_status = movingToRoombin
∧ (s_distance_robot_operator = safe ∨ s_distance_robot_operator = near))
→ change_speed_robot(limited)

(s_robot_position = room_bin ∧ s_robot_status = loading)
→change_speed_robot(stop)

(s_robot_position = workstation ∧ s_robot_status = dumping)
→ change_speed_robot(stop)

(s_robot_position = free_area ∧ s_distance_robot_operator = safe)
→ change_speed_robot(fast)

(s_robot_position = free_area ∧ s_distance_robot_operator = near)
→ change_speed_robot(limited)

s_distance_robot_operator = critical → change_speed(stop)

((s_robot_status = loading ∨ s_robot_status = dumping)
∧ s_distance_robot_operator = safe)
→ change_speed_arm(fast)

((s_robot_status = loading ∨ s_robot_status = dumping)
∧ s_distance_robot_operator = near ∧
(s_nearest_body_part = head ∨ s_nearest_body_part = hands)
→ change_speed_arm(limited)

((s_robot_status = loading ∨ s_robot_status = dumping)
∧ s_distance_robot_operator = critical ∧ s_nearest_body_part = head
→ change_speed_arm(stop)

((s_robot_status = loading ∨ s_robot_status = dumping)

∧ s_distance_robot_operator = critical ∧ s_nearest_body_part = hands

→ change_speed_arm(limited)

((s_robot_status = loading ∨ s_robot_status = dumping)

∧ s_distance_robot_operator = direct_contact

→ change_speed_arm(stop)

end Controller