Fuzzy Navigation Behaviors

AI and Robotics - Lab 03

Giacomo Gaiani - giacomo.gaiani@studio.unibo.it

Submission Date: 29/04/2024 Submission Number: 1

Contents

1 Task 1: Implementation of the "GoToTarget" behavior using fuzzy rule-based controller

1.1 Problem Description

The objective of all the tasks presented in the laboratory exercise is to define a set of fuzzy predicates, linguistic variables and fuzzy rules in order to implement a fuzzy navigation behavior for the simulated tiago robot. This is done by modifying or redesigning a python class defined in the `robot_gtw.py, observer.py` and `controller.py` files provided for the laboratory assignment [1]. In this first activity, the required task is to define the *GoToTarget* behavior that control the motion of the robot in order to reach the specified goal position.

1.2 Objects and Data Structures

All of the tasks presented in the laboratory require to define a subclass of an already defined python class called *Behavior*. Within these behavior classes, it is necessary to define the fuzzy predicates and rules to shape the actions that will be performed by the robot. The remaining objects and data structures are imported from the previous laboratories.

1.3 Control and Information Flow

The operational structure of the program can be divided in three sections. The first one is the data collection, namely: the wheel's encoders rotation, used to compute the robot position at each cycle; the information form the sonar sensors and the ground truth position of the robot, used to initialize the robot position and as a reference in the plotted trajectories. Contrary to the previous laboratory, only the five frontal sonar sensor are used positioned at 0,30,60,300 and 330 degree with respect to the robot reference system. This is done to make the data collection quicker and reduce the complexity of the relative fuzzy controllers. Only the computed position and the sensor data is passed to the controller. In the second section, the controller propagates the fuzzy predicates and rules defined

in the behavior class to compute the value of the linear and angular velocity. In the third section, the velocity values are passed to the robot via a rospy publisher. These three sections are executed each cycle until the maximum number of step in reached or a specific goal is achieved. In case of the *GoToTarget* behavior, the goal is to reach the target position, up to a certain tolerance.

1.4 Parameters

In order to clearly explain the implementation and tuning of the different navigation behaviors, I decided to split the current chapter in three different sections, analyzing separately the fuzzy predicates, the linguistic variables, and the fuzzy rules.

1.4.1 Fuzzy Predicates

The set of fuzzy predicates already provided was defined as:

```
'TargetLeft'  : (ramp_up(30.0, 60.0), 'phi'),
'TargetRight' : (ramp_down(-60.0, -30.0), 'phi'),
'TargetAhead' : (triangle(-60.0, 0.0, 60.0), 'phi'),
'TargetHere'  : (ramp_down(0.1, 2.0), 'rho'),
```

In order to smooth the trajectory of the robot, I added two more predicates:

```
'TargetLeftAhead'   : (triangle(0.0, 15.0, 30.0), 'phi'),
'TargetRightAhead'  : (triangle(-30.0, -15.0, 0.0), 'phi'),
```

The variables *rho* and *phi* represent respectively the linear and angular distance of the robot from the target position.

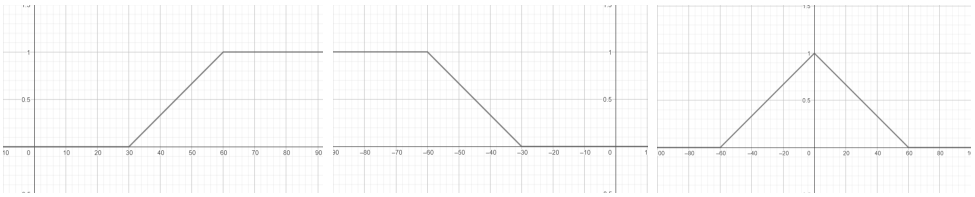


Figure 1: Fuzzy logic graph relative to the *TargetLeft* predicate

Figure 2: Fuzzy logic graph relative to the *TargetRight* predicate

Figure 3: Fuzzy logic graph relative to the *TargetAhead* predicate

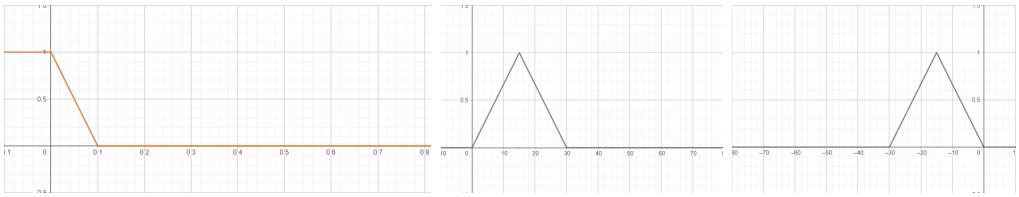


Figure 4: Fuzzy logic graph relative to the *TargetHere* predicate

Figure 5: Fuzzy logic graph relative to the *TargetMLeft* predicate

Figure 6: Fuzzy logic graph relative to the *TargetMRight* predicate

1.4.2 Linguistic Variables

The set of linguistic variables already provided was defined as:

```
'Move' : ({'Fast':0.5, 'Slow':0.1, 'None':0, 'Back':-0.1}, 'Vlin'),
'Turn' : ({'Left':40, 'MLeft':10, 'None':0, 'MRight':-10, 'Right':-40}, 'Vrot')
```

1.4.3 Fuzzy Rules

The set of fuzzy rules already provided was defined as:

```
'ToLeft'  : ("TargetLeft AND NOT(TargetHere)", 'Turn', 'Left'),
'ToRight' : ("TargetRight AND NOT(TargetHere)", 'Turn', 'Right'),
'Far'     : ("TargetAhead AND NOT(TargetHere)", 'Move', 'Fast'),
'Stop'    : ("TargetHere", 'Move', 'None')
```

Furthermore, I added two rules relative to the new predicates:

```
'ToMLeft' : ("TargetLeftAhead AND NOT(TargetHere)", 'Turn', 'MLeft'),
'ToMRight' : ("TargetRightAhead AND NOT(TargetHere)", 'Turn', 'MRight'),
```

The logic behind the fuzzy rules is to make the robot rotate in the direction of the target and move forward if the target is within an defined angle with respect to the direction the robot is facing.

1.5 Results

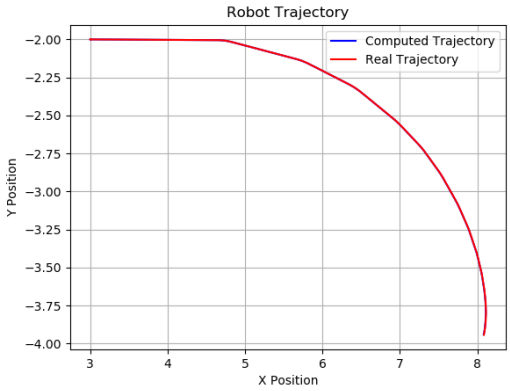


Figure 7: The robot trajectory using the default behavior

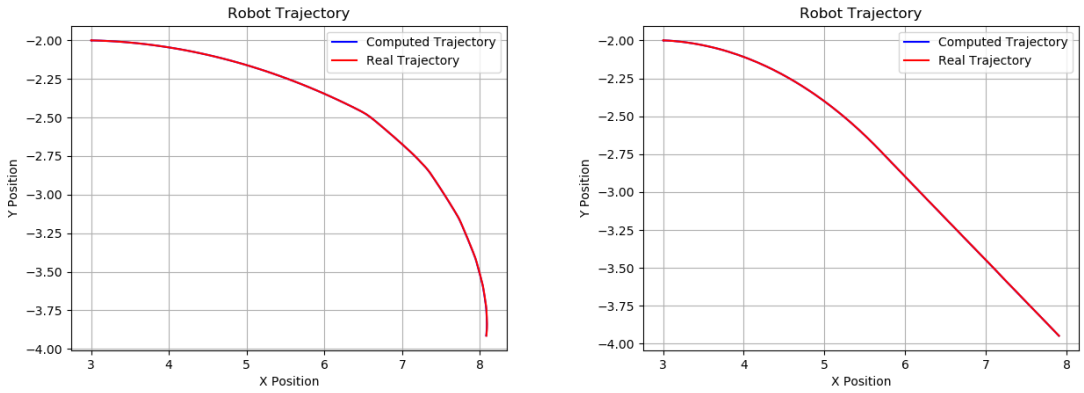


Figure 8: The robot trajectory using the additional predicates and MLeft = 2, MRight = -2

Figure 9: The robot trajectory using the additional predicates and MLeft = 5, MRight = -5

1.6 Conclusions

The figure 7 shows the trajectory of the robot using the *GoToTarget* behavior already defined for the laboratory activity. In order to implement a smoother navigation, the two new fuzzy predicates, presented in the Parameters chapter, has been added, while also lowering the angular velocity of the variables *MLeft* and *MRight*. In figures 8 and 9 is possible to the the resulting trajectories. The main difference with respect to the default one is the relative angle to with the robot start rotating. This helps the robot adjusting its position before being to close to the target, lowering the risk of overshooting the goal position.

2 Task 2: Implementation of the "Avoid" behavior using fuzzy rule-based controller

2.1 Problem Description

The objective of this second task is to define the *Avoid* behavior that control the motion of the robot in order to move it in a confined environment without suffering collisions.

2.2 Objects and Data Structures

The only difference with the previous task is the definition of *Avoid*, which is a new subclass of the already described *Behavior* class.

2.3 Control and Information Flow

The operational structure of the program is identical to the first task. The only difference is the stopping criteria, that does not involve a goal. Instead the program runs for a specified number of cycles.

2.4 Parameters

To solve the task, two set of fuzzy predicates and rules have been implemented. The first set define a behavior to simply avoid any collision, by moving away from any detected object. The second behavior make the robot cycle the room clockwise by maintaining a safe distance form the detected objects, following the perimeter of the room.

2.4.1 Fuzzy Predicates

The set of fuzzy predicates relative to the first behavior:

```
'ObjectAhead'       : (ramp_down(1.0, 3.0), 'sFront'),
'ObjectMLeft'       : (ramp_down(1.0, 2.0), 'sMLeft'),
```

```
'ObjectLeft'        : (ramp_down(1.0, 2.0), 'sLeft'),
'ObjectMRight'      : (ramp_down(1.0, 2.0), 'sMRight'),
'ObjectRight'       : (ramp_down(1.0, 2.0), 'sRight'),
```

The variables *sFront, sMLeft, sLeft, sMRight* and *sRight* represent the output of the five sonar sensors. To implement the second behavior, the following predicates have been added to the previous set:

```
'ObjectFarMLeft'    : (triangle(1.0, 2.0, 3.0), 'sMLeft'),
'ObjectFarLeft'     : (triangle(1.0, 2.0, 3.0), 'sLeft'),
'ObjectFarMRight'   : (triangle(1.0, 2.0, 3.0), 'sMRight'),
'ObjectFarRight'    : (triangle(1.0, 2.0, 3.0), 'sRight'),
```

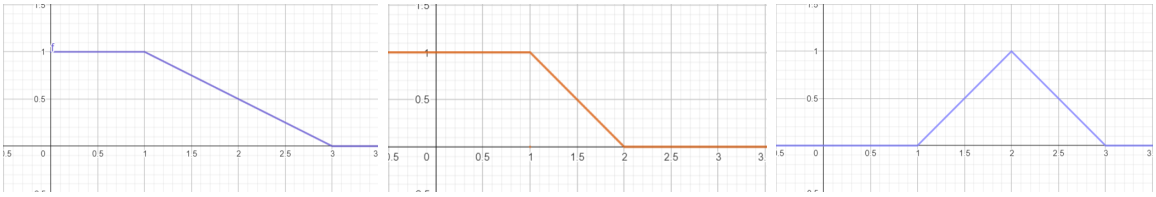


Figure 10: Fuzzy logic graph relative to the *ObjectAhead* predicate

Figure 11: Fuzzy logic graph relative to the *ObjectMLeft, ObjectLeft, ObjectMRight* and *ObjectRight* predicates

Figure 12: Fuzzy logic graph relative to the *ObjectFarMLeft, ObjectFarLeft, ObjectFarMRight* and *ObjectFarRight* predicates

With those additional predicates, the controller can discriminate between close and far objects.

2.4.2 Linguistic Variables

The set of linguistic variables are identical to the default *GoToTarget* class:

```
'Move' : ({'Fast':0.5, 'Slow':0.1, 'None':0, 'Back':-0.1}, 'Vlin'),
'Turn' : ({'Left':40, 'MLeft':10, 'None':0, 'MRight':-10, 'Right':-40}, 'Vrot')
```

2.4.3 Fuzzy Rules

The set of fuzzy rules relative to the first behavior is defined as:

```
'NoObjects': ("NOT(ObjectAhead) AND NOT(ObjectMLeft) AND NOT(ObjectLeft)
    AND NOT(ObjectMRight) AND NOT(ObjectRight)", 'Move', 'Fast'),
'AvoidLeft': ("ObjectLeft", 'Turn', 'MRight'),
'AvoidMLeft': ("ObjectMLeft", 'Turn', 'Right'),
'AvoidRight': ("ObjectRight", 'Turn', 'MLeft'),
'AvoidMRight': ("ObjectMRight", 'Turn', 'Left'),
```

The set of fuzzy rules relative to the second behavior is defined as:

```
'NoObjects': ("NOT(ObjectAhead) AND NOT(ObjectMLeft) AND NOT(ObjectMRight)",
    'Move', 'Fast'),
'AvoidLeft': ("ObjectLeft OR ObjectMLeft", 'Turn', 'MRight'),
'AvoidRight': ("ObjectRight OR ObjectMRight AND NOT(ObjectAhead)",
    'Turn', 'MLeft'),
'StayCloseLeft': ("ObjectFarLeft AND NOT(ObjectAhead)", 'Turn', 'MLeft'),
'AvoidInFornt': ("ObjectAhead", 'Turn', 'Right'),
```

The logic behind the fuzzy rules and the differences between the two behavior are explained in the "Conclusions" chapter.

2.5 Results

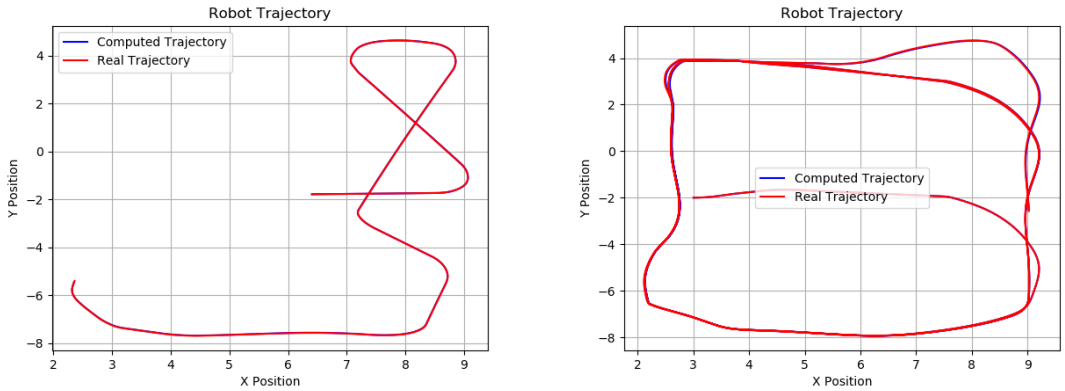


Figure 13: The robot trajectory using the simple avoid behavior, n° steps = 500

Figure 14: The robot trajectory using the "follow perimeter" avoid behavior, n° steps = 1800

2.6 Conclusions

The figure 13 shows the trajectory of the robot using the first *Avoid* behavior. It is possible to see how the robot simple "bounces" around the room, changing direction every time it detects an object along its path. The figure 14 shows the trajectory of the robot using the second *Avoid* behavior. It is possible to see how, after reaching a wall, it cycles around following the perimeter of the room. The main problem encountered during the implementation of this second behavior is to define the rules in such a way to prevent the robot from ending up stuck during the execution. To solve that, the defined set of rules is asymmetric, forcing the robot to turn right if stuck and to stay close to the leftmost detected object.

3 Task 3: Implementation of the "FollowObject" behavior using fuzzy rule-based controller

3.1 Problem Description

The objective of this third task is to define the *FollowObject* behavior that control the motion of the robot in order to move it in the direction of the closest detected object up to a certain distance. If the object is closer than this defined distance, the robot will move away from it.

3.2 Objects and Data Structures

The only difference with the previous task is the definition of *FollowObject*, which is a new subclass of the already described *Behavior* class.

3.3 Control and Information Flow

The operational structure of the program and the stopping criteria are identical to the second task.

3.4 Parameters

3.4.1 Fuzzy Predicates

The set of fuzzy predicates defined to solve the task are identical to the first set described for the *Avoid* class, with the addition of the predicate *ObjectTooClose*:

```
'ObjectAhead'       : (ramp_down(2.0, 3.0), 'sFront'),
'ObjectTooClose'    : (ramp_down(1.0, 1.5), 'sFront'),
'ObjectMLeft'       : (ramp_down(1.0, 2.0), 'sMLeft'),
'ObjectLeft'        : (ramp_down(1.0, 2.0), 'sLeft'),
'ObjectMRight'      : (ramp_down(1.0, 2.0), 'sMRight'),
'ObjectRight'       : (ramp_down(1.0, 2.0), 'sRight'),
```

The variables *sFront, sMLeft, sLeft, sMRight* and *sRight* represent the output of the five sonar sensors.

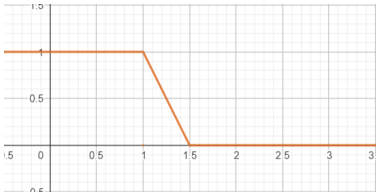


Figure 15: Fuzzy logic graph relative to the *ObjectTooClose* predicate

With this additional predicates, the controller can discriminate if the object the robot is heading to is too close.

3.4.2 Linguistic Variables

The set of linguistic variables are identical to the default *GoToTarget* class:

```
'Move' : ({'Fast':0.5, 'Slow':0.1, 'None':0, 'Back':-0.1}, 'Vlin'),
'Turn' : ({'Left':40, 'MLeft':10, 'None':0, 'MRight':-10, 'Right':-40}, 'Vrot')
```

3.4.3 Fuzzy Rules

The set of fuzzy rules defined for this behavior is:

```
'NoObjects': ("NOT(ObjectAhead) AND NOT(ObjectMLeft) AND NOT(ObjectMRight)",
    'Move', 'Fast'),
'FollowInFront': ("ObjectAhead AND NOT(ObjectTooClose)", 'Move', 'Fast'),
'FollowLeft': ("ObjectLeft OR ObjectMLeft", 'Turn', 'Left'),
'FollowRight': ("ObjectRight OR ObjectMRight", 'Turn', 'Right'),
'TooClose' : ("ObjectTooClose", 'Move', 'Back')
```

The logic behind the fuzzy rules is to make the robot rotate towards the closest object and move it (forward or backward) to reach a certain distance.

3.5 Results

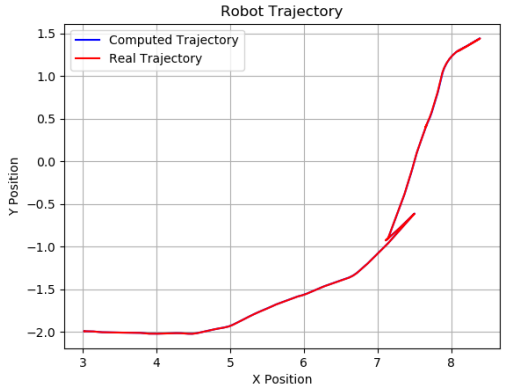


Figure 16: The robot trajectory using the *FollowObject* behavior, n° steps = 200

3.6 Conclusions

The figure 16 shows the trajectory of the robot using the *FollowObject* behavior. In order to perform the test, a cube has been placed in front of the robot and then moved in a series of positions. In the trajectory it is possible to see when the cube has been moved closer to the robot, making it move backwards to maintain a safety distance.

4 Task 4: Implementation of the "CrossDoor" behavior using fuzzy rule-based controller

4.1 Problem Description

The objective of this last task is to define the *CrossDoor* behavior that control the motion of the robot in order to make it cross a door close to its starting position.

4.2 Objects and Data Structures

The only difference with the previous task is the definition of *CrossDoor*, which is a new subclass of the already described *Behavior* class.

4.3 Control and Information Flow

The operational structure of the program is identical to the first task. The only difference is the stopping criteria. The program runs until the door is completely crossed.

4.4 Parameters

4.4.1 Fuzzy Predicates

The set of fuzzy predicates defined to solve the task are:

```
'ObjectAhead'       : (ramp_down(1.0, 3.0), 'sFront'),
'ObjectMLeft'       : (ramp_down(1.0, 3.0), 'sMLeft'),
'ObjectLeft'        : (ramp_down(1.0, 3.0), 'sLeft'),
'ObjectMRight'      : (ramp_down(1.0, 3.0), 'sMRight'),
'ObjectRight'       : (ramp_down(1.0, 3.0), 'sRight'),
'ObjectAround'      : (ramp_down(2.0, 3.0), 'sAround'),
```

The variables *sFront*, *sMLeft*, *sLeft*, *sMRight* and *sRight* represent the output of the five sonar sensors, while *sAround* is their average. This last predicate is used as stopping mechanism to interpret if the door is fully corssed.

4.4.2 Linguistic Variables

The set of linguistic variables are identical to the default *GoToTarget* class:

```
'Move' : ({'Fast':0.5, 'Slow':0.1, 'None':0, 'Back':-0.1}, 'Vlin'),
'Turn' : ({'Left':40, 'MLeft':10, 'None':0, 'MRight':-10, 'Right':-40}, 'Vrot')
```

4.4.3 Fuzzy Rules

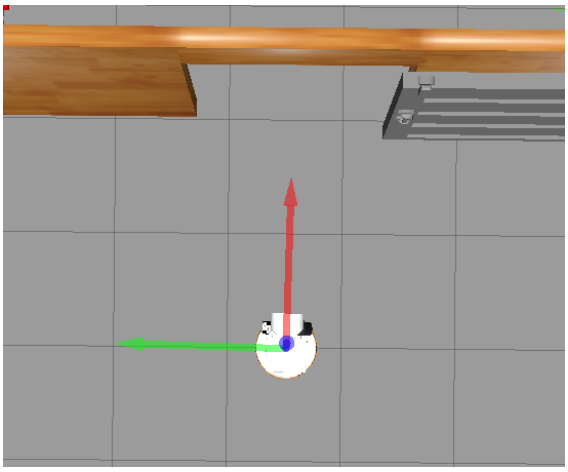
The set of fuzzy rules defined for this behavior is:

```
'FacingDoor': ("NOT(ObjectAhead)", 'Move', 'Fast'),
'PassedDoor': ("NOT(ObjectAround)", 'Move', 'None'),
'DoorOnLeft': ("NOT(ObjectLeft)", 'Turn', 'MLeft'),
'DoorOnRight': ("NOT(ObjectRight)", 'Turn', 'MRight'),
'DoorOnMLeft': ("NOT(ObjectMLeft)", 'Turn', 'Left'),
'DoorOnMRight': ("NOT(ObjectMRight)", 'Turn', 'Right'),
```

The logic behind the fuzzy rules is that if a sensor does not detect an object, it is because it is probably where the door is.

4.5 Results

In order to evaluate the performance of the *CrossDoor* behavior, I have performed three tests using the same parameters but different conditions. In the first test, the robot was set the axis perpendicular to the center of the door, facing in that direction. In the second test the robot was set on the same position, but with an angular offset. In the third test the robot was set parallel to the axis perpendicular to the center of the door. For this last test, the space on the other side of the door, with respect to the robot, has been closed.



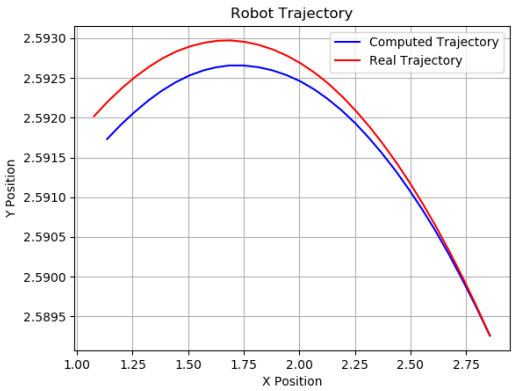
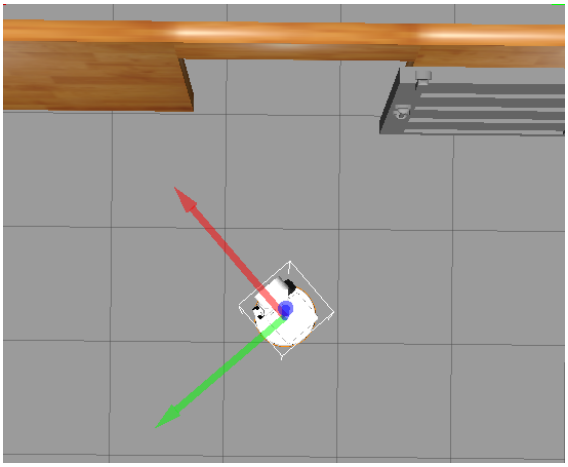


Figure 17: The starting position of the first test

Figure 18: The robot trajectory using the *CrossDoor* behavior



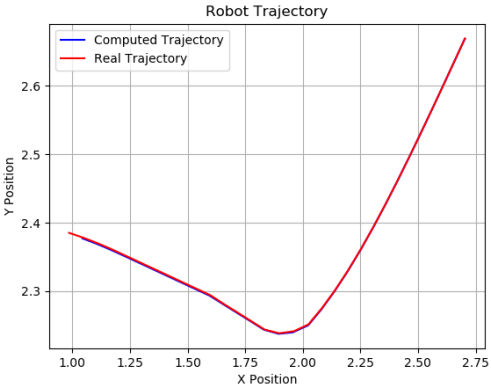
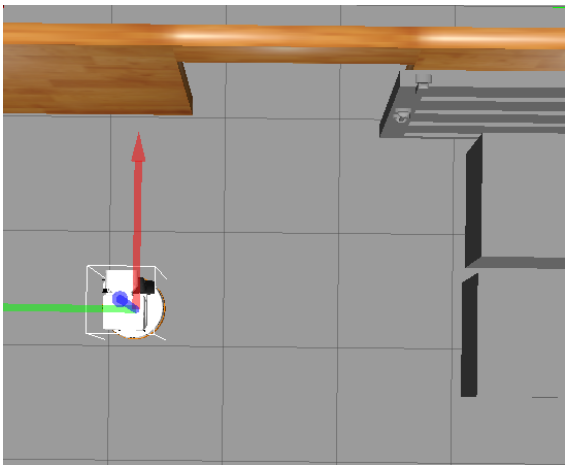


Figure 19: The starting position of the second test

Figure 20: The robot trajectory using the *CrossDoor* behavior



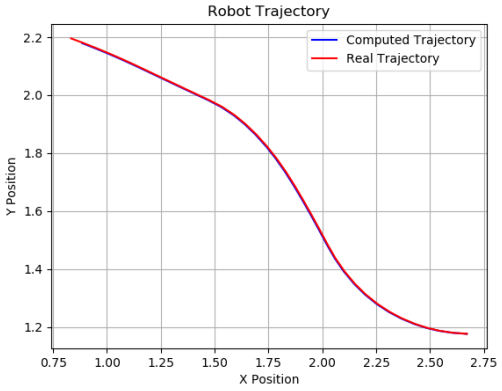


Figure 21: The starting position of the third test

Figure 22: The robot trajectory using the *CrossDoor* behavior

4.6 Conclusions

The figure 17 shows the resulting trajectory of the first test using the *CrossDoor* behavior. The robot moved in a strait line until completely crossing the door. The figure 20 shows the resulting trajectory of the second test using the *CrossDoor* behavior. The robot started moving while still not be precisely in the direction of the door but adjusted the position throughout the time needed to cross it. The figure 22 shows the resulting trajectory of the third test using the *CrossDoor* behavior. The robot performed an "S" shaped trajectory to adjust for the initial offset. If the test was performed without blocking the other side of the door, the robot would have just followed the wall, disregarding the door. A simple solution for this problem would be to give the robot the information about the position and orientation of the door.

References

[1] Alessando Saffiotti. *BaseCode-2024*. 2024. URL: https://github.com/asaffio/BaseCode-2024.