



## The project

**Automation of reverse parking  
using rules and logic, regardless of  
the size of the trailer  
and its tractor.**

# The Problems

- 1) The mathematics for the dynamic of a multiple axles trailer being pushed-back
- 2) The skill of driving
- 3) The algorithm to always find a parking maneuver, whatever the configuration



# Methods used to solve the aforementioned problems

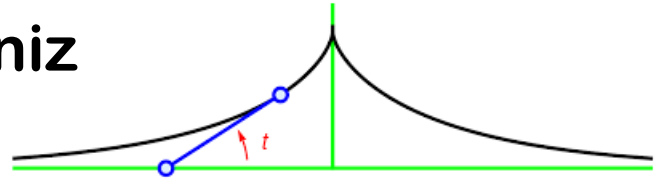
- 1) High-Frequency sampling to discretize the trajectory
- 2) Control-Loops
- 3) State-Machine





# Chassis Simulation

1) The tractrix of Leibniz

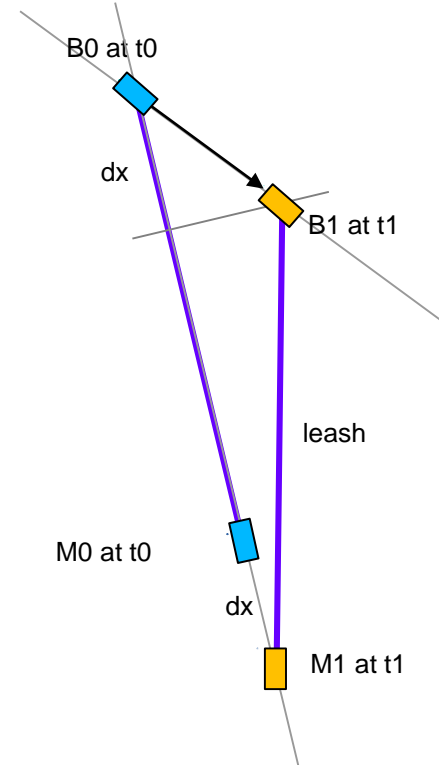
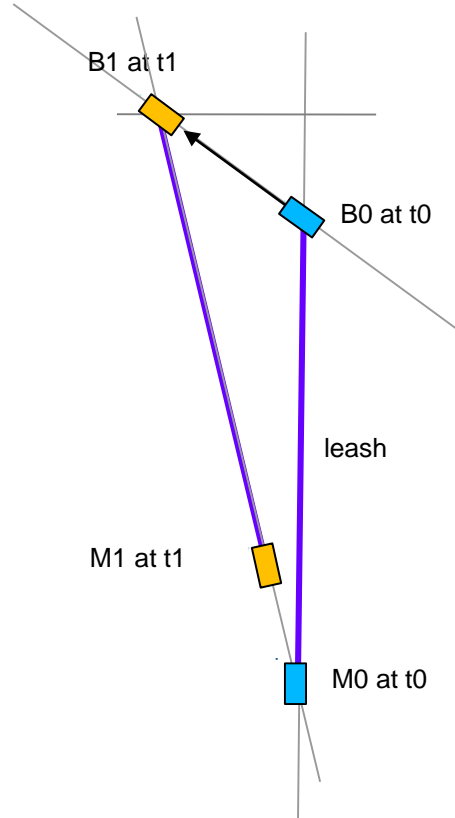


2) Mathematical formulation

3) High-Frequency sampling to discretize the tractory



# Tractory discretization





## High-Freq Sampling pros

- Basic Newton's laws & vector algebra
- Angle based control-loops including user defined delays
- Well adapted to real-time drawing and checking

## High-Freq Sampling cons

- Needs 30 Hz or above for a better accuracy of the trajectory approximation
- Short term prediction only
- Needs a real-time engine

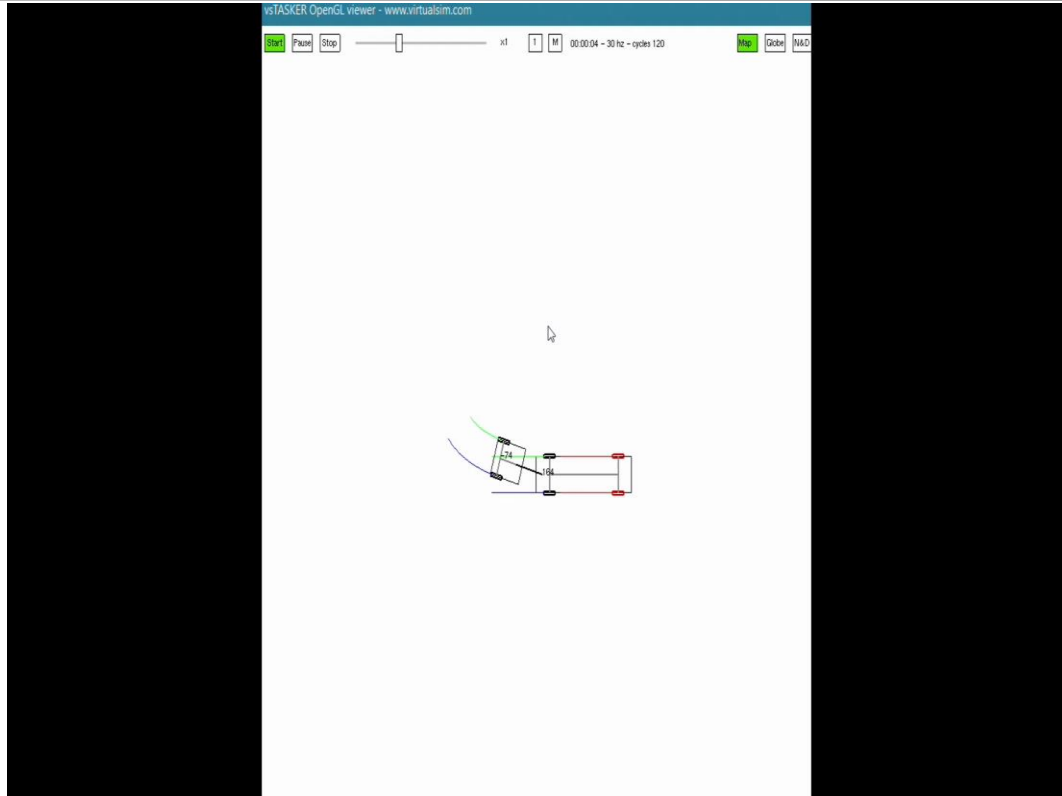




14-16 May 2019  
Stockholmsmässan, Sweden



# Validation





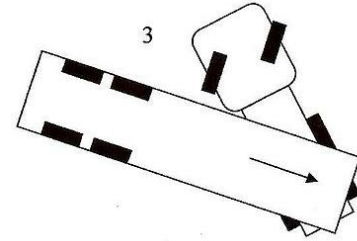


# Driving Skills

- Steering the driving wheels
- Getting & Keeping the hitch angle



# Jackknife Angle

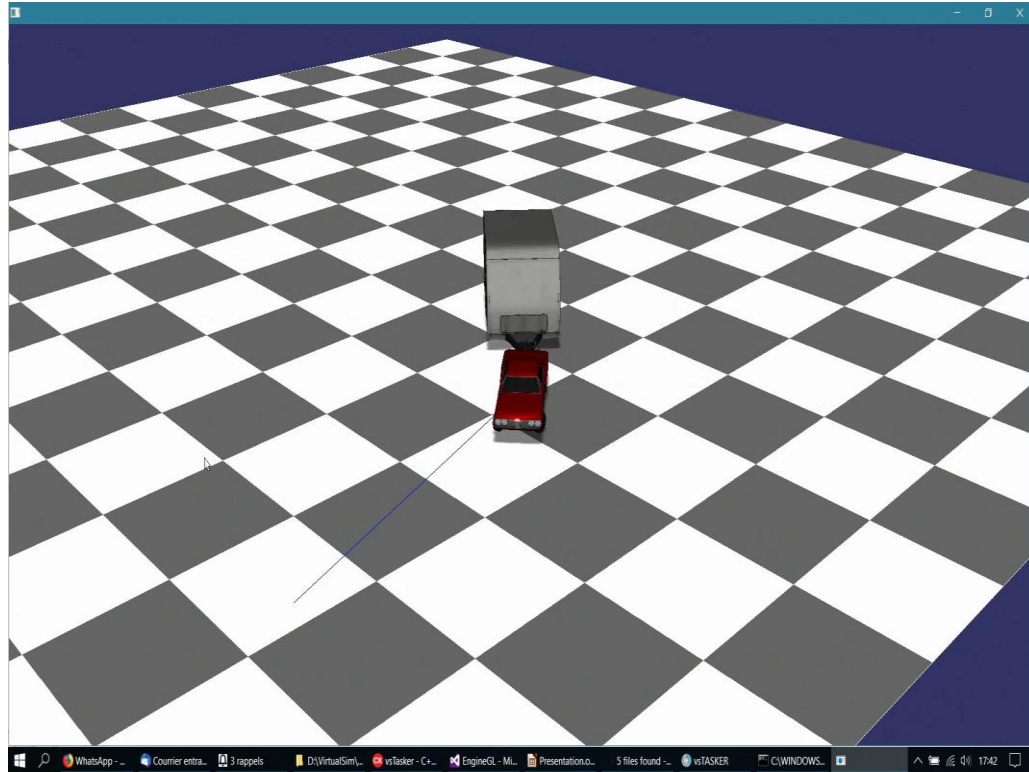


- The jackknife problem
- Empirically finding the jackknife angle
- Computing the minimum turn radius in forward and reverse modes

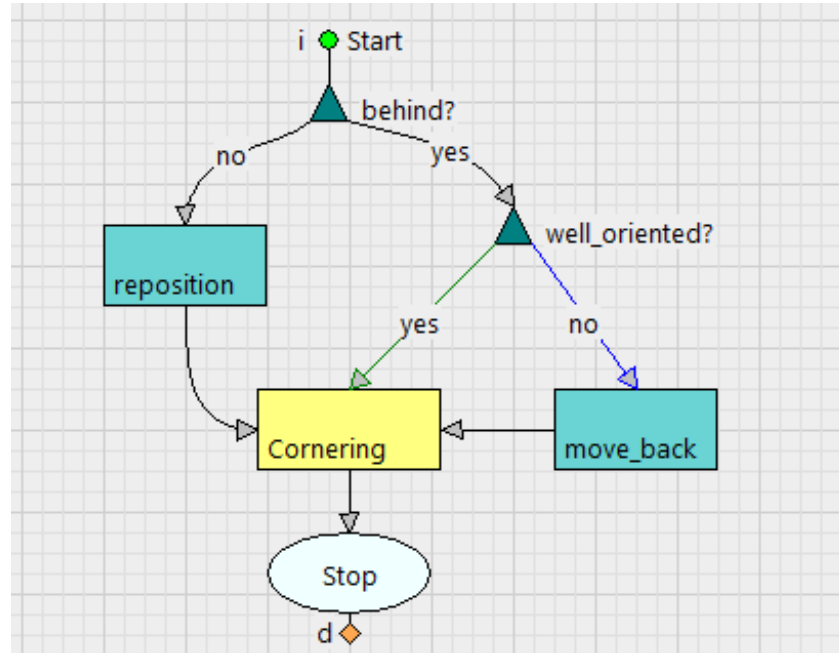




# Jackknife Resolution

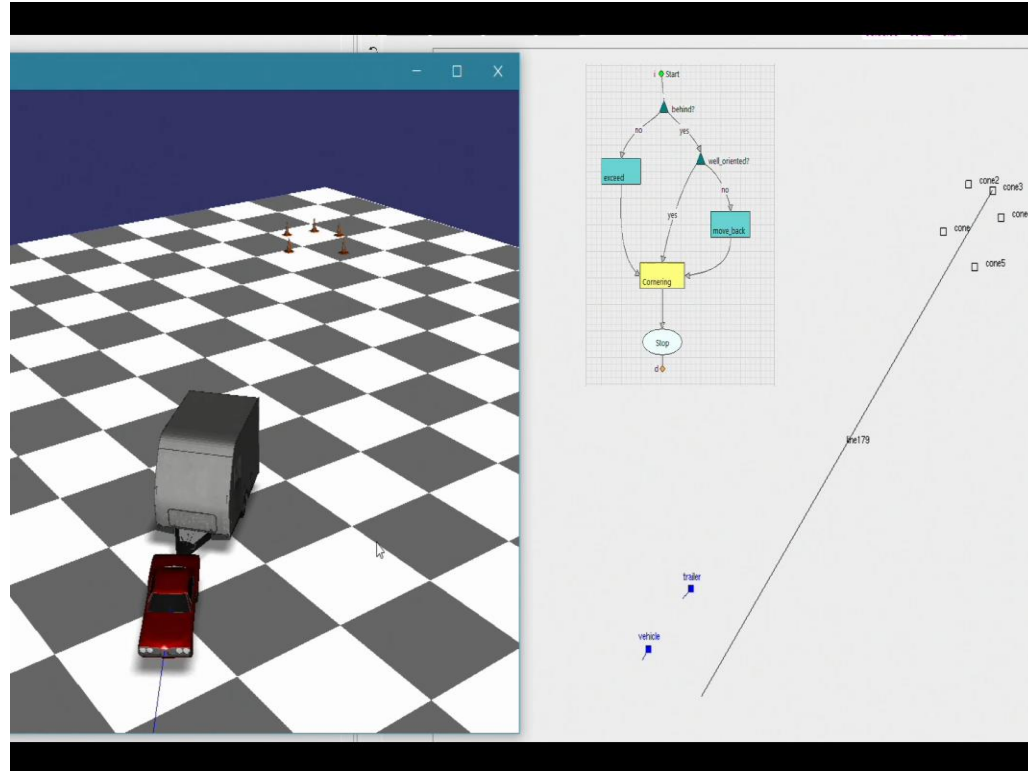


# Maneuver Logic





# Maneuver Test



The screenshot displays a simulation environment with a 3D scene on the left and a logic flowchart on the right. The 3D scene shows a red car pulling a grey trailer on a black and white checkered floor. A mouse cursor is visible over the floor. The logic flowchart is a state machine with the following elements:

- Start** (green circle)
- behind?** (green diamond decision node)
- waited** (cyan rectangle action node)
- wait\_oriented?** (green diamond decision node)
- move\_back** (cyan rectangle action node)
- Comparing** (yellow rectangle action node)
- Stop** (white oval terminal node)

The flowchart logic is as follows:

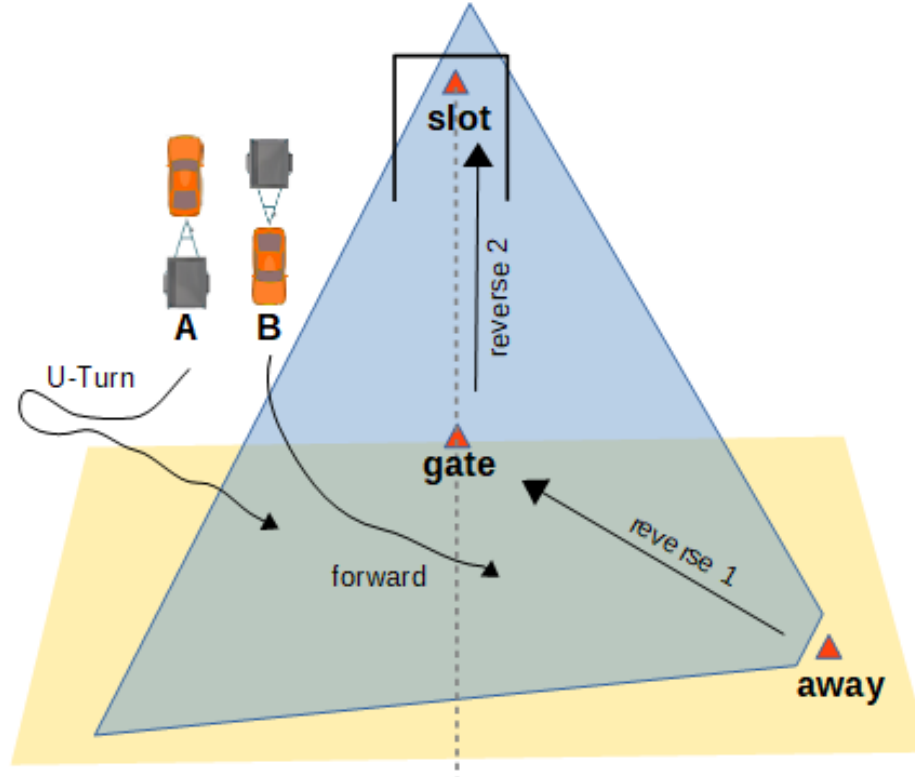
```

graph TD
    Start((Start)) --> Behind{behind?}
    Behind -- no --> Waited[waited]
    Behind -- yes --> WaitOriented{wait_oriented?}
    WaitOriented -- yes --> MoveBack[move_back]
    WaitOriented -- no --> Comparing[Comparing]
    MoveBack --> Comparing
    Waited --> Comparing
    Comparing --> Stop((Stop))
  
```

On the right side of the simulation interface, there are several objects labeled: cone2, cone3, cone4, cone5, cone, and a line labeled line179. At the bottom, there are labels for 'trailer' and 'vehicle' with blue arrows pointing to their respective 3D models.



# The 3 points Algorithm



# Realignment

**Sometimes the cornering will fail aligning the trailer correctly after the Gate point;**

**Then, need to realign by moving forward;**

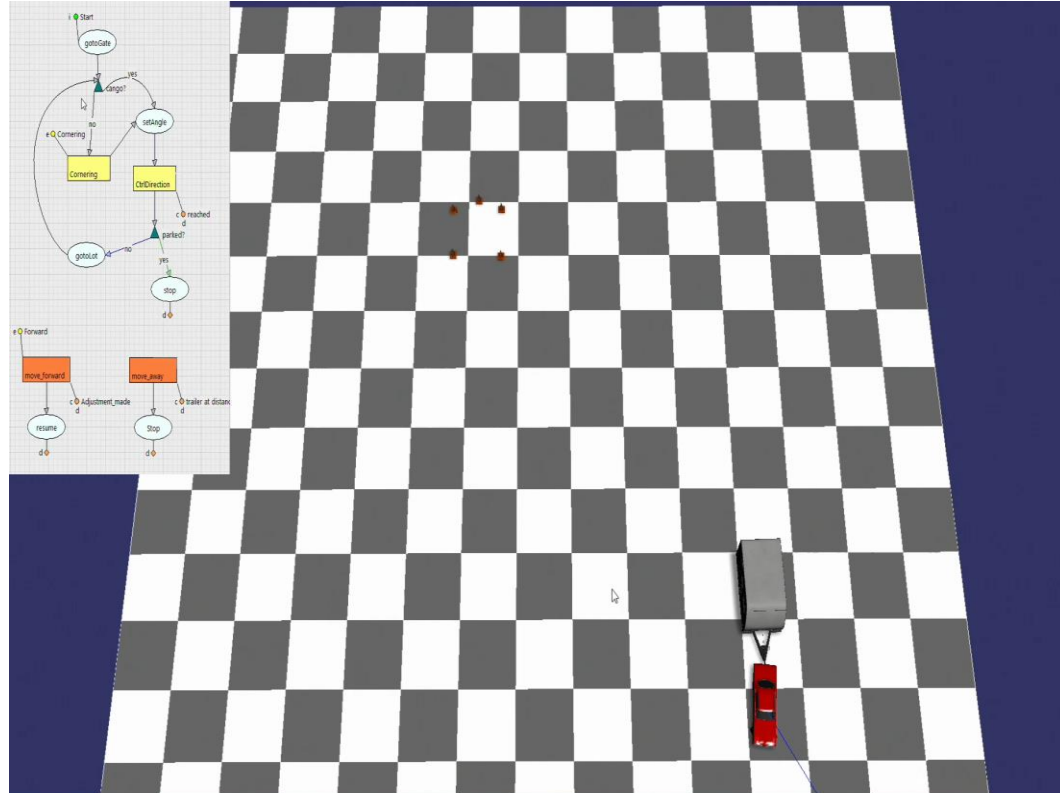
**Automatic triggering from the state-machine.**







## Short trailer parking

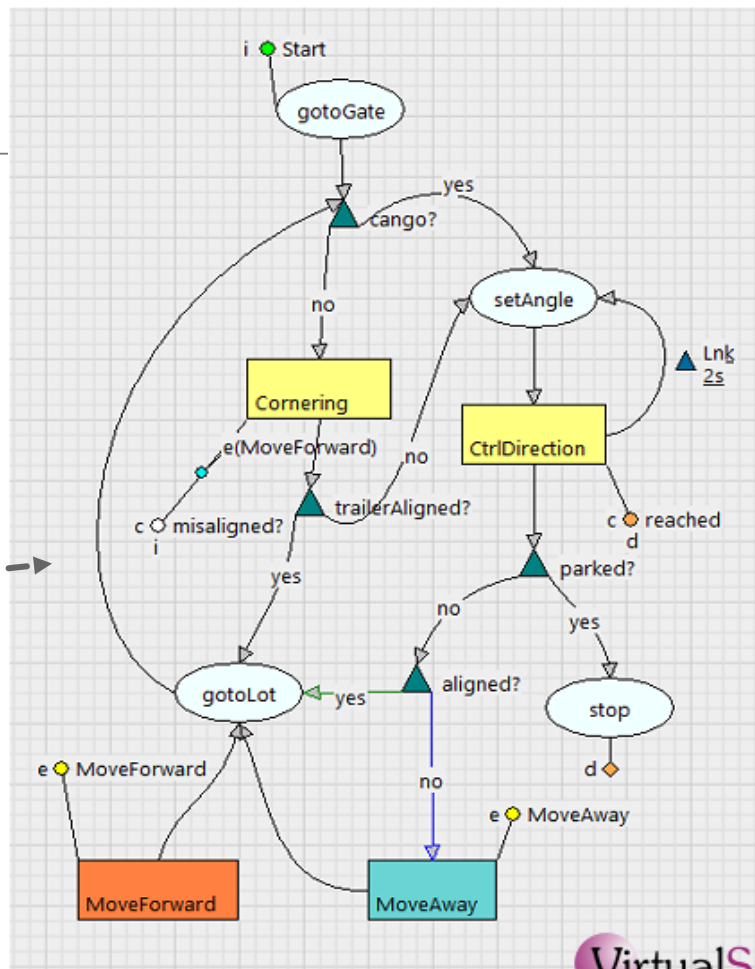
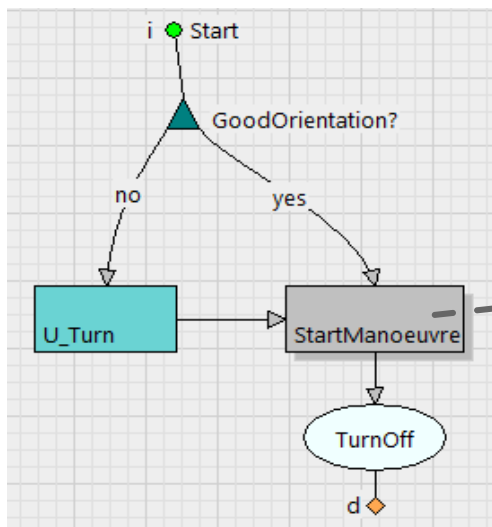


The diagram is a state machine for a short trailer parking task. It starts at a 'Start' node leading to a 'gotoGate' state. A decision diamond 'Cargo?' branches to 'yes' (leading to 'setAngle') and 'no' (leading to 'Cornering'). 'Cornering' leads to 'getOut' and back to 'Cargo?'. 'setAngle' leads to 'OutDirection'. A decision diamond 'reached d' branches to 'yes' (leading to 'stop') and 'no' (leading to 'paralel?'). 'paralel?' leads to 'stop'. A second decision diamond 'trailer at distan' branches to 'yes' (leading to 'stop') and 'no' (leading to 'move\_forward'). 'move\_forward' leads to 'Adjustment\_made', which leads to 'resume'. 'move\_forward' also leads to 'move\_back' (highlighted in orange), which leads to 'stop'.



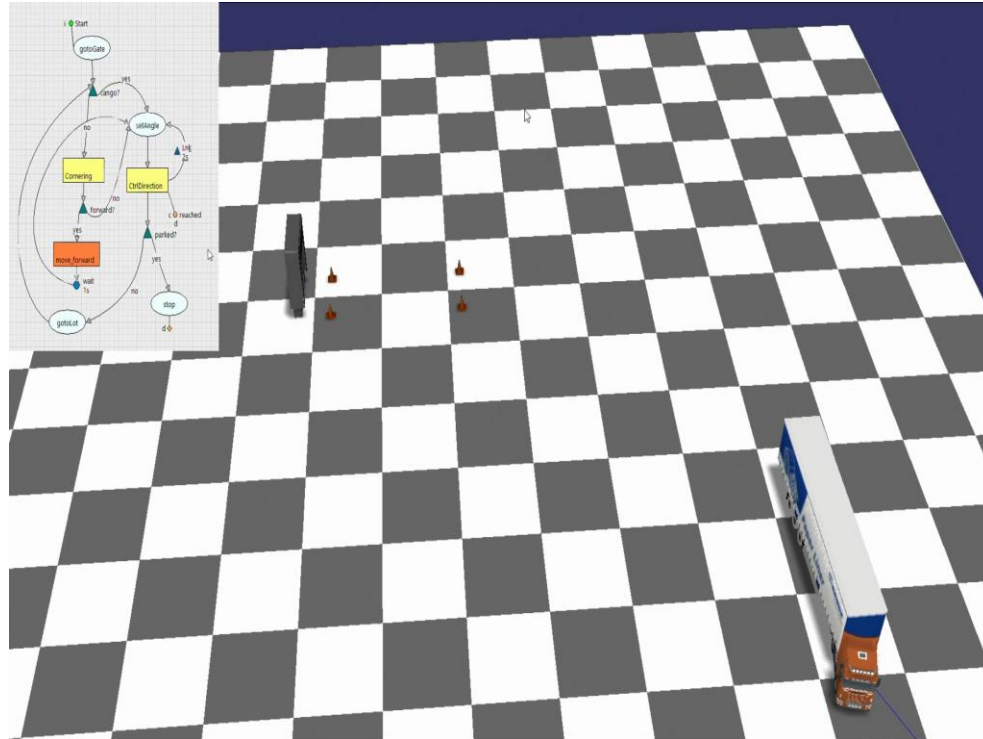


## Final State-Machine



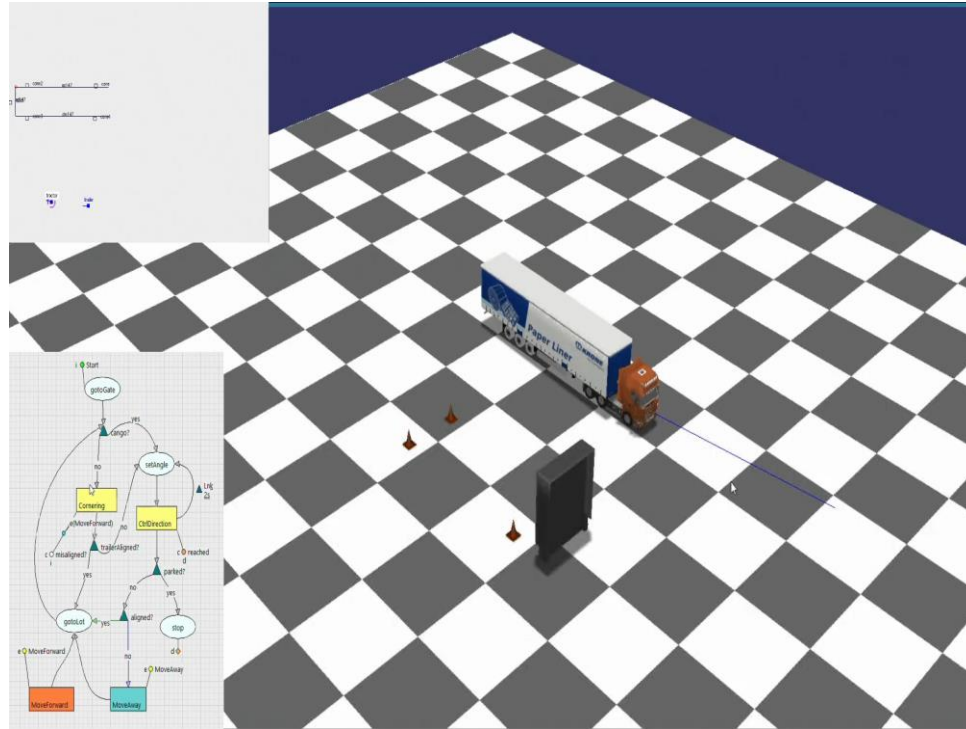


# Long Trailer Parking 1





## Long Trailer Parking 2



The diagram is a state machine for the truck's parking task. It starts at a 'Start' node leading to 'getToGate'. From 'getToGate', the truck moves forward ('e0 MoveForward') to a 'change' node. From 'change', it can either 'align' (if 'c0 misaligned?') or 'rotate' (if 'c0 rotated?'). The 'align' path leads to a 'Cornering' state, which involves 'MoveForward' and 'MoveKeyway' actions, leading to a 'getToC' node. The 'rotate' path leads to a 'rotate' state, which involves 'MoveKeyway' and 'MoveForward' actions, leading to a 'getToC' node. From 'getToC', the truck can 'stop' (if 'c0 aligned?') or 'align' (if 'c0 misaligned?'). The 'stop' state leads to a 'stop' node, which then leads to a 'stop' state. The 'align' path leads to a 'stop' node, which then leads to a 'stop' state. The 'stop' state leads to a 'stop' node, which then leads to a 'stop' state.

## Conclusion

**Discretization provided the same accuracy as complex maths.**

**Basic control loops allowed precise alignments and turns.**

**A single state-machine was enough to solve most configurations.**



**Thank-you !**

**Any  
Question ?**

