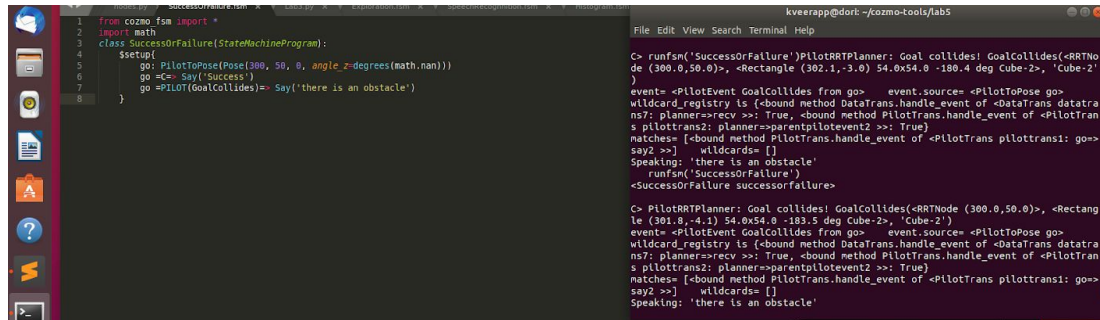


SuccessOrFailure:

For this problem, we used the PilotToPose command to steer Cozmo to that location. Using a completion transition and a PILOT transition, we cased on whether there was an obstacle in Cozmo's way, displaying appropriate messages in `simpe_cli` if there was a success or obstacle. Images of the results are displayed below.

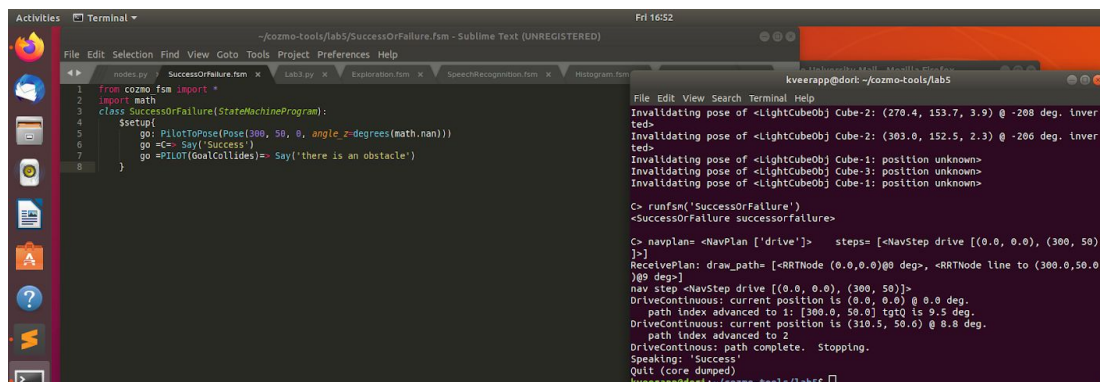


```
1 from cozmo_fsm import *
2 import math
3 class SuccessOrFailure(StateMachineProgram):
4     $setup{
5         go: PilotToPose(Pose(300, 50, 0, angle_z=degrees(math.nan)))
6         go =<= Say('Success')
7         go =>PILOT(GoalCollides)=> Say('there is an obstacle')
8     }

C> runfsm('SuccessOrFailure')PilotRRTPPlanner: Goal collides! GoalCollides(<RTNode (300.0,50.0)>, <Rectangle (302.1,-3.0) 54.0x54.0 -180.4 deg Cube-2>, 'Cube-2' )
events=<PilotEvent GoalCollides from go> event-source=<PilotToPose go>
wildcard_registry is {<bound method DataTrans.handle_event of <DataTrans dataTrans7: planner=>recv >>: True, <bound method PilotTrans.handle_event of <PilotTrans pilotTrans2: planner=>parentPilotEvent2 >>: True}>
matches= [<bound method PilotTrans.handle_event of <PilotTrans pilotTrans1: go> say2 >>] wildcards= []
Speaking: 'there is an obstacle'
runfsm('SuccessOrFailure')
<SuccessOrFailure successorfailure>

C> PilotRRTPPlanner: Goal collides! GoalCollides(<RTNode (300.0,50.0)>, <Rectangle (301.8,-4.1) 54.0x54.0 -183.5 deg Cube-2>, 'Cube-2')
events=<PilotEvent GoalCollides from go> event-source=<PilotToPose go>
wildcard_registry is {<bound method DataTrans.handle_event of <DataTrans dataTrans7: planner=>recv >>: True, <bound method PilotTrans.handle_event of <PilotTrans pilotTrans2: planner=>parentPilotEvent2 >>: True}>
matches= [<bound method PilotTrans.handle_event of <PilotTrans pilotTrans1: go> say2 >>] wildcards= []
Speaking: 'there is an obstacle'
```

Cozmo says the text 'There is an obstacle' when cubes were obstructing the destination.



```
1 from cozmo_fsm import *
2 import math
3 class SuccessOrFailure(StateMachineProgram):
4     $setup{
5         go: PilotToPose(Pose(300, 50, 0, angle_z=degrees(math.nan)))
6         go =<= Say('Success')
7         go =>PILOT(GoalCollides)=> Say('there is an obstacle')
8     }

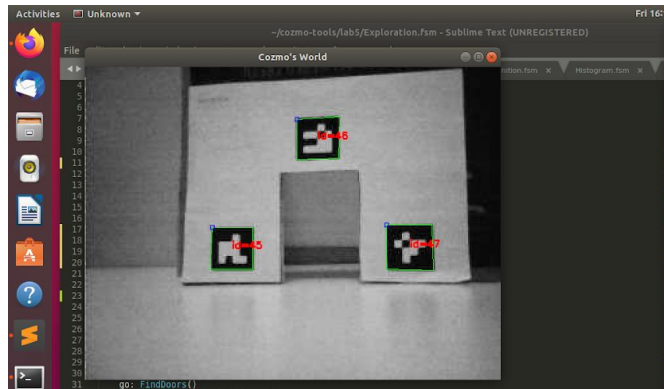
C> runfsm('SuccessOrFailure')
<SuccessOrFailure successorfailure>

C> navplan=<NavPlan ['drive']> steps= [<NavStep drive [(0.0, 0.0), (300, 50)]> ]
ReceivePlan: draw_path= [<RTNode (0.0,0.0)@0 deg>, <RTNode line to (300.0,50.0)@0 deg>]
nav step=<NavStep drive [(0.0, 0.0), (300, 50)]>
DriveContinuous: current position is (0.0, 0.0) @ 0.0 deg.
path index advanced to 1: [300.0, 50.0] tgtQ is 9.5 deg.
DriveContinuous: current position is (310.5, 50.0) @ 0.0 deg.
path index advanced to 2
DriveContinuous: path complete. Stopping.
Speaking: 'Success'
Quit (core dumped)
kveerapp@dort:~/cozmo-tools/lab5
```

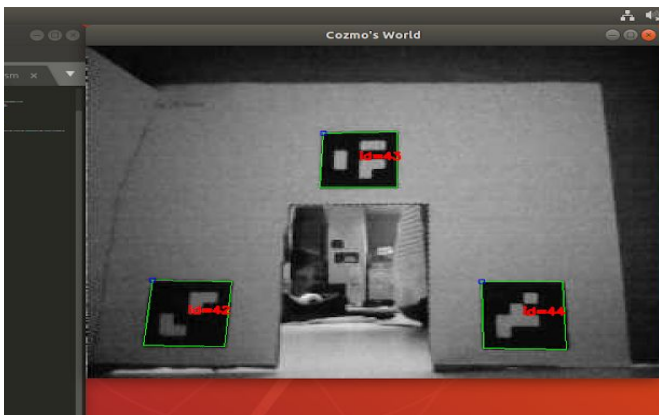
Cozmo says 'Success' and moves to the destination when there is no obstacle.

Exploration:

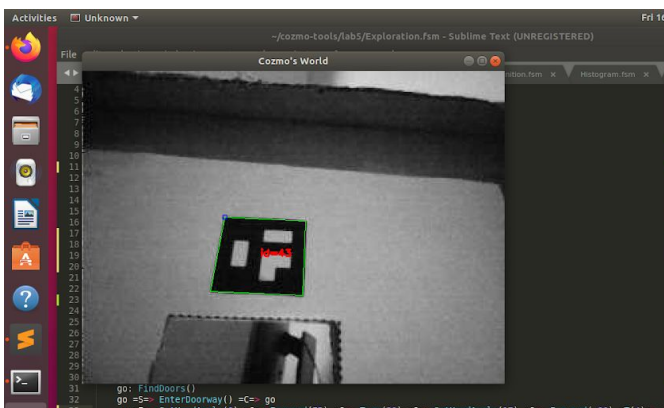
We used a State Node and a DoorPass Node for this problem. In the state node, we looped through the key-value pairs of `robot.world.world_map.objects.items()` and, if the keyword 'Doorway-' was in the key and if this key was not already in the globally defined list of doors Cozmo has already passed through. In this case, the class posted a success. Else, the class posts a failure. In the case that the FindDoors class returns a success, Cozmo enters the EnterDoorway class and passes through the door. If Cozmo does not find a door, he goes through a series of forward, backward, turn, and head angle turns to check for more doorways. Cozmo does not go through the same doorway twice. Below are some images of Cozmo navigating through his shack, which has two doors.



Cozmo sees three ArUco markers and recognizes that there is a doorway outside of the Shack.



Cozmo sees two ArUco markers once inside of the shack, knows that he has not seen this doorway before, and orients himself to enter the doorway.



Cozmo is now passing through the doorway he previously identified.