As simulações foram implementadas com o sistema operacional, ferramentas e pacotes indicados a seguir:

- Ubuntu 20.04 LTS arm64 (virtualizado no Parallels em computadores com chip Apple M1)
- ROS 2 Foxy Fitzroy
- Gazebo 11
- Python 3.8.10
- CasADi v3.6.4
- HSL Solver – versão de 2015 (opcional)

1. Instalação das ferramentas no Ubuntu

a) Instalar o ROS 2 seguindo as instruções deste link.
b) Instalar o Colcon seguindo as instruções deste link.
c) Instalar o Gazebo seguindo as instruções deste link.
d) Instalar os pacotes que permitem a integração do ROS 2 e Gazebo seguindo as instruções deste link.
e) Instalar o CasADi para Python seguindo as instruções deste link.
f) Instalar o HSL Solver seguindo as instruções deste link. É necessário fazer a solicitação e o link para download será disponibilizado. O uso do HSL é opcional, entretanto recomendável para acelerar as iterações do MPC.

2. Preparação dos pacotes na borda

a) Abra um terminal no diretório /edge disponibilizado e execute o comando:

```
colcon build
```

b) Existem três pacotes na borda: “edge_tier_pkg”, “edge_launch” e “interfaces_pkg”. Esses pacotes serão construídos após a execução do colcon. Caso queira modificar os códigos dos dois primeiros pacotes sem a necessidade de executar novamente o colcon, ainda no diretório /edge, faça:

```
colcon build --packages-select edge_tier_pkg --symlink-install

colcon build --packages-select edge_launch --symlink-install
```

c) Para que os comandos do ROS 2 se tornem acessíveis, execute no terminal:

```
echo "source /opt/ros/foxy/setup.bash" >> ~/.bashrc
```

É necessário executar o comando acima uma única vez após a instalação do ROS 2. Veja mais detalhes neste link.

d) Para que os pacotes construídos também se tornem acessíveis, execute o comando abaixo, colocando o caminho até o diretório /edge:

```
echo "source /caminho_ate_o_diretorio/edge/install/setup.bash" >>
~/.bashrc
```

3. Preparação dos pacotes no lado local

a) Nas simulações realizadas para os resultados da tese, os pacotes da borda e do lado local foram executados em dois computadores diferentes, conectados em rede cabeada. Para a preparação dos pacotes no lado local, a mesma sequência de comandos deve ser executada, portanto, no mesmo ou em outro computador, a partir do diretório /local:

```
colcon build

colcon build --packages-select local_tier_pkg --symlink-install

colcon build --packages-select local_launch --symlink-install

echo "source /caminho_ate_o_diretorio/local/install/setup.bash" >>
~/.bashrc
```

4. Habilitação do Solver MA57

a) Caso tenha instalado o HSL Solver, habilite o MA57 nos nós de MPC da borda e do lado local. Abra os arquivos mpc_node.py e mpc_tracking_node.py nos pacotes edge_tier_pkg e local_tier_pkg, respectivamente. Procure a variável ma57_solver no construtor da classe MPC para a borda e da classe MPCLocalNode para o lado local. Altere o valor lógico dessas variáveis para True.

5. Inicialização dos nós

a) A demonstração apresentada neste documento será baseada no Cenário 1, sendo o mesmo procedimento para os demais. No computador que representa a borda, faça:

```
ros2 launch edge_launch edge.scn_1.launch.py
```

Esse launch file irá executar quatro nós de MPC para planejamento da trajetória, dado os quatro AGVs do cenário, além do nó do Supervisor e sua interface gráfica. Caso deseje executar o nó do Supervisor em outro terminal, desabilite as linhas de código do respectivo launch file que inicializam o nó do Supervisor e execute no outro terminal:

```
ros2 run edge_tier_pkg supervisor
```

b) No lado local são executados dois launch files. O primeiro executará o Gazebo e lançará os componentes que compõem o cenário, como AGVs e obstáculos. O segundo executará os quatro nós do MPC de rastreamento da trajetória. Em terminais diferentes, execute cada um dos comandos a seguir:

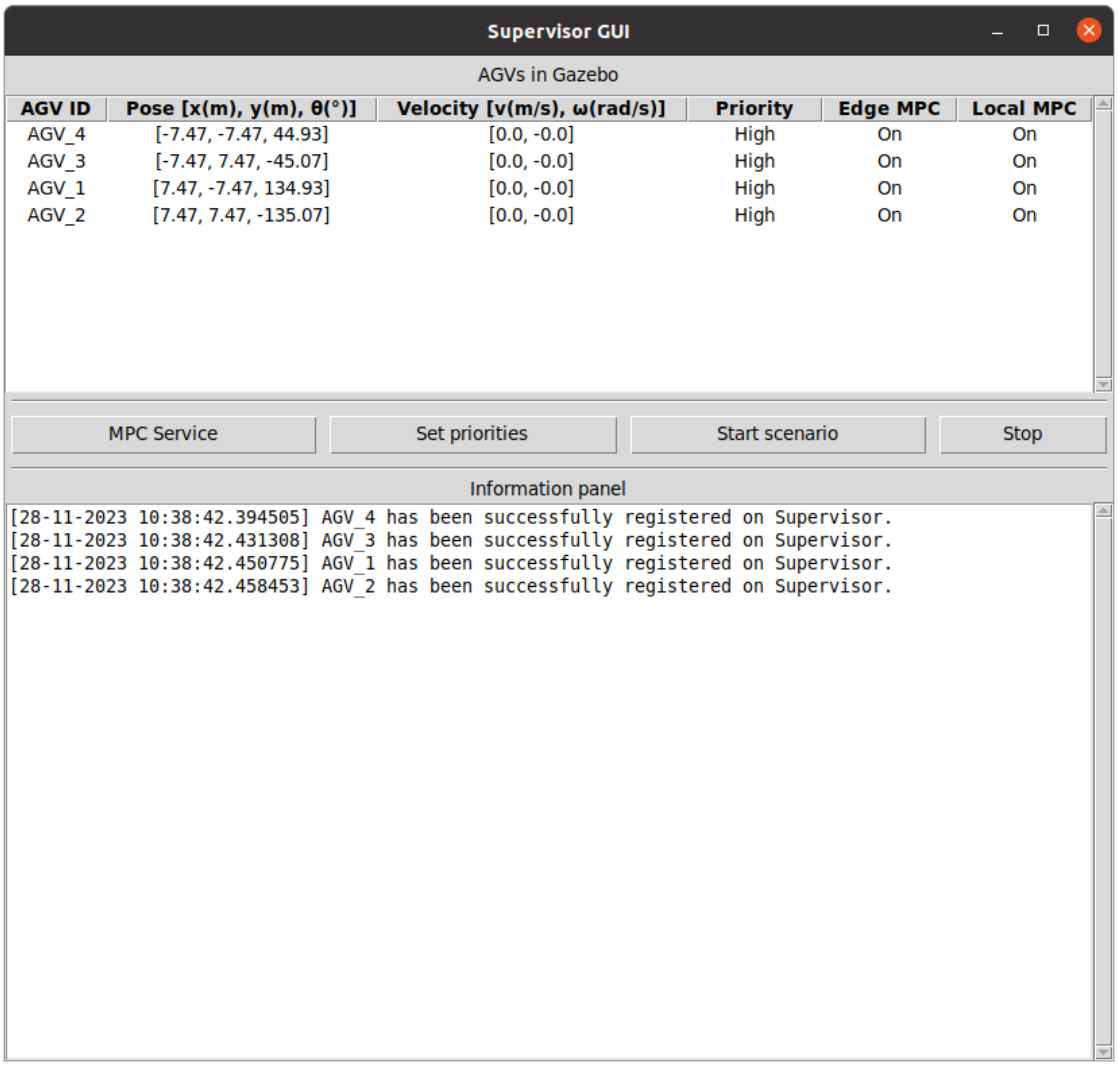
```
ros2 launch local_launch scn_1.launch.py
```

```
ros2 launch local_launch local_1_all.launch.py
```

Se preferir, instale o Terminator e tenha vários terminais em uma única janela. Detalhes da instalação estão neste link.

Nas simulações para os resultados da tese foi utilizada uma máquina virtual para cada AGV no computador do lado local, de modo a representar o computador embarcado no AGV de uma aplicação real. No caso do Cenário 1, um total de cinco máquinas virtuais foram utilizadas, sendo uma de cada AGV e uma para o Gazebo. Nessa situação foi criado um launch file para cada máquina virtual, o qual executava um nó de MPC para o respectivo AGV.

6. Interface gráfica do Supervisor



a) Ao carregar todos os nós do cenário, os dados dos AGVs devem ser vistos na interface gráfica e o cenário está pronto para ser iniciado.

b) No botão ‘MPC Service’ é possível que o serviço de MPC seja chamado individualmente para cada AGV do cenário. Na janela que abrirá ao clicar

no botão, é possível informar os dados de posição para onde o AGV deve ir e chamar o serviço.

c) No botão ‘Set priorities’ é possível ajustar a prioridade de cada AGV, podendo ser alta (‘High’), média (‘Medium’) ou baixa (‘Low’). Neste trabalho a prioridade apenas determina a velocidade linear máxima do AGV. Em projetos futuros outras funções podem ser adicionadas no Supervisor, como por exemplo, em uma situação de conflito, o AGV com maior prioridade possa ter preferência na passagem.

d) Ao clicar no botão ‘Start scenario’ o Supervisor irá iniciar os procedimentos que estão previstos para o cenário em questão. No cenário 1 o Supervisor irá chamar o serviço de MPC para cada AGV, de modo que os veículos troquem de posição na diagonal. Nos cenários 2 e 3, a posição dos AGVs é controlada pelo Supervisor para atender a aplicação prevista para o cenário. Nesse caso, os nós de MPC serão chamados diversas vezes até que a simulação seja interrompida.

e) Ao clicar no botão ‘Stop’ os AGVs estacionam onde estiverem naquele instante.

7. Inicialização dos outros cenários

a) Inicialmente é preciso alterar o código do arquivo supervisor.py do pacote edge_tier_pkg. No arquivo altere a classe que será herdada pela classe SupervisorNode, de acordo com o cenário. Nestas classes estão os algoritmos de supervisão para cada cenário. Para o cenário 2, por exemplo, a classe SupervisorNode herda as classes Node e Scenario2, conforme a seguir:

```
class SupervisorNode(Node, Scenario2):
```

Obs.: Caso os pacotes não tenham sido construídos com o --symlink-install, será necessário construí-los novamente para que a alteração no código tenha efeito.

b) Para inicializar os nós dos outros cenários, apenas altere o número do comando do launch file. Por exemplo, para o cenário 2, os comandos da borda e do lado local são os seguintes:

```
ros2 launch edge_launch edge.scn_2.launch.py

ros2 launch local_launch scn_2.launch.py

ros2 launch local_launch local_2_all.launch.py
```

8. Salvando dados da simulação

a) Se desejar salvar os dados da simulação, nos launch files da borda e do lado local altere o valor lógico da variável save_sim_data para True.

b) Na variável sim_name dos dois launch files coloque um nome que irá compor os nomes dos arquivos a serem salvos e que identifique a simulação.

c) Agora abra os arquivos mpc_node.py e mpc_tracking_node.py dos pacotes edge_tier_pkg e local_tier_pkg, respectivamente, e coloque na variável self.path o caminho onde os arquivos dos dados da simulação serão salvos.

d) No lado da borda serão salvos um arquivo de texto com o tempo de processamento do MPC em cada iteração e um arquivo texto com as métricas de uso do processador e memória. No lado local, além desses dois arquivos, serão salvos os arquivos de texto com os dados de trajetória percorrida pelo AGV e os sinais de controle enviados ao Gazebo.

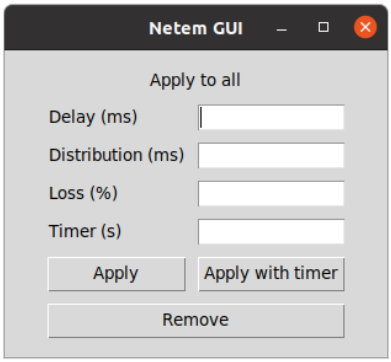
9. Aplicando atraso e perda de pacotes com o NetEm

a) Foi analisado também o comportamento da arquitetura proposta mediante uma rede degradada com atraso e perda de pacotes. Para isso foi utilizado o emulador de rede do Linux, o NetEm. Foram desenvolvidos também nós do ROS 2 nos dois lados, para execução dos comandos do NetEm e uma interface gráfica para interação com esses nós. Esses nós foram pensados para execução em máquinas virtuais individuais para cada AGV e para o Gazebo, de modo que a degradação é aplicada apenas na comunicação entre os nós locais e a borda, sem interferência na comunicação entre esses nós e o Gazebo. Pode ocorrer problemas caso os nós locais sejam executados na mesma máquina que o Gazebo.

b) Caso deseje utilizar o NetEm, altere o valor lógico da variável netem_node_enable para True, nos launch files da borda e local. Isso habilitará os nós do NetEm em ambos os lados. No lado local isso deve ser feito tanto no launch file que executa os nós de MPC (local_x_all.launch.py) como no launch file que lança o cenário (scn_x.launch.py). Este último habilita a interface gráfica para interação com os nós.

c) No código do arquivo netem_node.py do pacote local_tier_pkg, coloque o endereço IP da borda na variável self.edge_ip. O endereço é necessário para o filtro utilizado, de maneira que a degradação seja aplicada apenas na comunicação com a borda, sem interferência na comunicação com o Gazebo (caso executado em uma máquina virtual exclusiva).

d) Com o NetEm habilitado, a interface da figura a seguir será carregada. Em ‘Delay (ms)’ informe o atraso em milissegundos, em ‘Distribution’ a Distribuição Normal do atraso em milissegundos e em ‘Loss’ a porcentagem da perda de pacotes. Sempre preencha o campo ‘Loss’, mesmo que 0, caso a perda de pacotes desejada seja nula. Após o preenchimento dos campos, clique em ‘Apply’. A partir desse momento o NetEm estará ativado para os pacotes que deixam o lado local e que deixam a borda. Clique em ‘Remove’ para desabilitar o NetEm.



e) O NetEm também pode permanecer ativo apenas em um intervalo de tempo determinado. Para isso, além dos outros campos preencha o campo ‘Timer’ com um valor de tempo em segundos e clique em ‘Apply with timer’.

10. Parâmetros

a) Vários parâmetros são passados aos nós através dos launch files. Na borda, as variáveis que definem alguns desses parâmetros, são:

- agv_type: 1 para o AGV menor (Cenários 1, 2 e 4) e 2 para o AGV maior (Cenário 3)
- d_safe: distância de segurança entre os AGVs
- d_safe_obs: distância de segurança entre os AGVs e obstáculos fixos
- high_vel, medium_vel e low_vel: velocidades lineares em m/s para cada prioridade
- obstacles: quantidade dos obstáculos mais próximos a ser considerada pelo MPC na prevenção de colisões
- limit_n: quantidade dos AGVs mais próximos a ser considerada pelo MPC na prevenção de colisões

Esses dois últimos parâmetros podem ser utilizados para a diminuição do intervalo de tempo das iterações do MPC na borda. Quanto menor o número de obstáculos e AGVs a serem considerados, menor o intervalo de tempo da iteração. Quando o limit_n recebe o valor 0, todos os AGVs serão considerados e é o caso que demandará mais uso de CPU. O ajuste do limit_n, principalmente, é útil em simulações em que a quantidade de AGVs poderá colocar a CPU em seu limite máximo e tornar a simulação lenta. Reduzir a quantidade de AGVs a serem considerados na prevenção de colisões, portanto, aliviará o uso de CPU. Nas simulações para os resultados da tese, o limit_n foi ajustado para diferente de 0 nos Cenários 2 e 4. Esses cenários têm mais AGVs e ainda trafegam uns próximos aos outros, dificultando a obtenção da solução do MPC quanto à prevenção de colisões.

b) Alguns parâmetros ajustados nos launch files do lado local são também as prioridades e as velocidades. Em especial no Cenário 4, defina a quantidade de AGVs a serem lançados na simulação. Isso é feito através da variável n nos launch files local_4_all.launch.py e scn_4.launch.py. Nas simulações para os resultados da tese foram utilizados 12 AGVs.

11. Documentos relacionados

- Artigo CBA 2022
- Artigo IoT
- Tese