MPC Exercise

November 1, 2020

In this exercise you are supposed to apply MPC to make the TurtleBot3 follow a predefined reference (first in simulation and then in the lab). All necessary information to complete this task is given below.

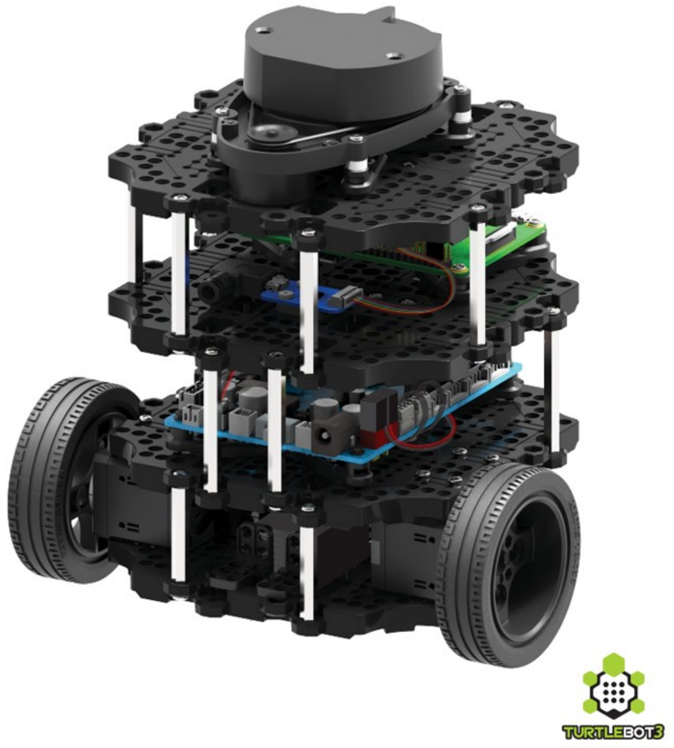


Figure 1: TurtleBot3

1 Specifications

The Turtblebot3 has the following Hardware specifications:

Maximum translational velocity (or speed)	0.22 m/s
Maximum Payload	15 Kg
Weight	1Kg

Table 1: Specifications for the Turtlebot3

The robot is desired to follow a circular trajectory defined as following:

$$ref_x(k) = -0.8 + 0.5\cos(0.2kT + \pi/4) \tag{1a}$$
$$ref_y(k) = -0.4 + 0.5\sin(0.2kT + \pi/4) \tag{1b}$$

Where T is the sample time, and $k \in \mathbb{Z}_{\geq 0}$. However, the robot is not allowed to cross the line $y = -0.6$ while moving on the circular trajectory as shown in the figure below:

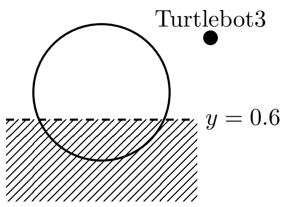


Figure 2: Circular Trajectory of the robot with the unwanted region

The kinematics of the Turtlebot3 can be modeled as following:

$$x(k+1) = x(k) + v(k)\cos(\theta(k))T \tag{2a}$$
$$y(k+1) = y(k) + v(k)\sin(\theta(k))T \tag{2b}$$
$$\theta(k+1) = \theta(k) + \omega(k)T \tag{2c}$$

Where:

x : is the x position of the robot in the inertial frame $\mathcal{I}$.

y : is the y position of the robot in the inertial frame $\mathcal{I}$.

v : is the velocity (or speed) of the robot in its body frame $\mathcal{B}$.

θ : is the heading angle of the robot (Rotation angle between the inertial frame to the body frame around the z axis.)

ω : is the angular velocity of the heading.

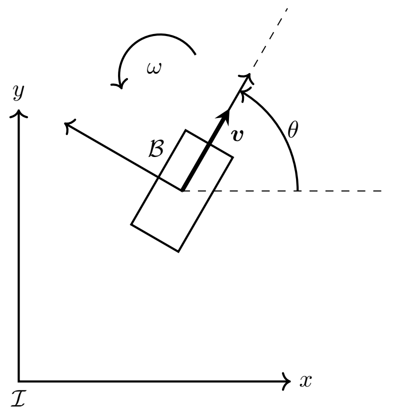


Figure 3: The body frame of the robot in the inertial frame with translational velocity vector $\boldsymbol{v}$ and angular velocity ω.

The model in (2) is nonlinear and, therefore, a linear MPC cannot be implemented. To deal with this, the MPC will only be relative to the following position model:

$$x(k+1) = x(k) + Tv_x(k) \tag{3a}$$
$$y(k+1) = y(k) + Tv_y(k) \tag{3b}$$
$$\tag{3c}$$

Where:

$$v_x(k) = v(k)\cos(\theta(k))$$
$$v_y(k) = v(k)\sin(\theta(k))$$

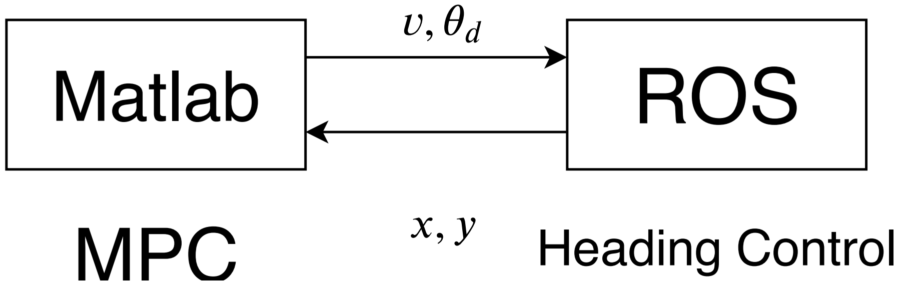
Note that v_x and v_y here are just the component of the velocity vector $\boldsymbol{v}$ in the inertial frame. The MPC will then use this model to predict and optimize for the control variables v_x, v_y. After obtaining the control variables v_x and v_y, the velocity in (2) can be recovered from the velocity vector $\boldsymbol{v}$ as following:

$$v = |\boldsymbol{v}| = \sqrt{v_x^2 + v_y^2} \tag{4}$$

and a desired heading angle can be computed from:

$$\theta_d = \text{atan2}(v_y, v_x) \tag{5}$$

Afterwards, a simple P controller running at a higher rate on the robot can control the heading to the desired angle. The following figure shows the structure of the setup:



The MPC will be running at a lower sample time than the simulation and ROS during implementation. The input to the robot from the MPC between its samples is constant, this is also known as zero-order-hold. The figure below illustrate the concept:

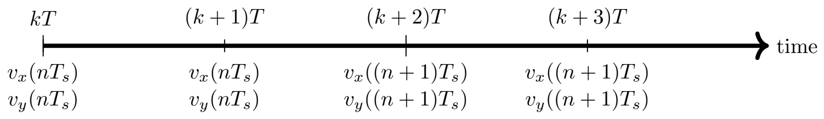


Figure 4: The MPC is running at a sample time T_s lower than the simulation/ROS sample time T. The simulation/ROS apply the same input it has from the MPC until it gets a new one.

Five Matlab files are provided:

- `Config.m`: Configuration script to define some parameters.
- `MPCdefinition.m`: the script which should contain the optimizer-object **MPCobj**, presented in Section 2.
- `MPCsimPredictionModel.m`: simulation script where the linear model is used as plant.
- `MPCsim.m`: simulation script where the non-linear model is used as plant.
- `MPCros.m`: implementation script that communicates with a ROS master on the Turtlebot3.

2 Simulation Exercise

In this exercise, your task is to specify a Model Predictive Control (MPC) strategy for the specifications given in the previous section, and implement it as a YALMIP optimizer-object, called **MPCobj**. The link to the opimizer-object documentation is found here: https://yalmip.github.io/command/optimizer/.

The YALMIP optimizer-object **MPCobj** takes the the current state $p(k) \in \mathbb{R}^2$, last input $u(k-1) \in \mathbb{R}^2$ and the reference in the prediction window $p_{ref} \in \mathbb{R}^{2\times H_p}$ as input, and provides prediction window containing the optimal states $p^* \in \mathbb{R}^{2\times H_p}$ and the control window containing the optimal inputs $u^* \in \mathbb{R}^{2\times H_u}$ as output.

Inputs

$$p(k) = \begin{bmatrix} x(k) & y(k) \end{bmatrix}^{\mathrm{T}} \tag{6a}$$

$$u(k-1) = \begin{bmatrix} v_x(k-1) & v_y(k-1) \end{bmatrix}^{\mathrm{T}} \tag{6b}$$

$$p_{ref} = \begin{bmatrix} x_{ref}(k+1) & ... & x_{ref}(k+H_p) \\ y_{ref}(k+1) & ... & y_{ref}(k+H_p) \end{bmatrix} \tag{6c}$$

$$\tag{6d}$$

Outputs

$$u^* = \begin{bmatrix} v_x(k) & v_y(k) & ... & v_x(k+H_u-1) & v_y(k+H_u-1) \end{bmatrix}^{\mathrm{T}} \tag{7a}$$

$$p^* = \begin{bmatrix} x(k+1) & y(k+1) & ... & x(k+H_p) & y(k+H_p) \end{bmatrix}^{\mathrm{T}} \tag{7b}$$

More compactly formulated

$$\textbf{MPCobj} : \{p(k), u(k-1), p_{ref}\} \mapsto \{u^*, p^*\} \tag{8}$$

The Yalmip object **MPCobj** is already defined in the file **MPCdefinition.m** with the necessary inputs and outputs.

We suggest you approach the problem in the following way

1. Choose a sample time T_s, prediction horizon H_p and control horizon H_u.
2. Define a suitable objective/cost function.
3. Formulate the appropriate constraints.
4. Use Δu to introduce integral action.
5. Lift the system.
6. Familiarise yourself with the Yalmip syntax.
7. Implement the solution the in the file **MPCdefinition.m**.

8. Use the simulation to adjust the penalty gains to improve the performance of the controller.

9. Play around with the script.

When you have defined the optimizer object according to the specifications, you can run the simulation by executing the file **MPCsimPredictionModel.m** or **MPCsim.m** in Matlab. The simulation in **MPCsimPredictionModel.m** uses the linear model, and in **MPCsim.m** the model is non-linear. Note that both simulations use the same configuration file **Config.m** for comparison reason.

Enjoy :)

3 ROS exercise

At this point you hopefully could get the MPC controller **MPCobj** to work with the simulations. Now it is time to try the controller on TurtleBot3. This will be done simply by executing the script **MPCros.m** in Matlab.

The script will use the object **MPCobj** to run the MPC controller directly on the robot. The script **MPCros.m** will not work without the Matlab toolbox **ROS toolbox**. Please install it before we go to the lab.