Model Predictive Control with Viriato

Yasmin Hesham
Faculty of Engineering
Alexandria University
yasmin98hesham@gmail.com

Pablo Bustos
RoboLab, School of Technology
University of Extremadura
Cáceres, 10003
pbustos@unex.es

April 25, 2022

Abstract

Study of Model Predictive Control in social-robotics scenarios.

1 Introduction

2 Viriato's kinematic model

Viriato is a four wheeled omni-directional robot that uses Mecanum wheels. It can move independently on the plane, whether forward, sideways and rotating. We start by deriving the kinematic model of Viriato in two steps. First the robot to world's frame relationship, and then the wheels to robot's frame relationship. The concatenation of both gives the complete model.

The kinematic model that relates the robot's frame to the world's frame is non-linear and is derived as follows. The state variables of Viriato are its 2D position in space and the rotation angle with respect to the Z axis, measured CCW from the Y axis: $x : \{x, y, \omega\}$. Viriato can move independently in these dimensions: forward, sideways and rotating. These three velocities are related to the corresponding velocities in world's reference system C as:

$$\begin{bmatrix} \dot{x_C} \\ \dot{y_C} \\ \dot{\omega_C} \end{bmatrix} = \begin{bmatrix} cos(\omega) & -sin(\omega) & 0 \\ sin(\omega) & cons(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\omega_R} \end{bmatrix} \tag{1}$$

This equation says that the velocity of the robot in the world's frame is a rotated version of the velocity of the robot in its own frame. Position is not represented here and we will have to integrate this equation to get to it.

Expression (1) is a non linear differential equation that can be re-written as:

$$\dot{x} = f(x, u) \tag{2}$$

where u is the vector of control inputs: $u : \{\dot{x_R}, \dot{y_R}, \dot{\omega_R}\}$ that change the state of the robot.

However, in order to use MPC on-line it would be very convenient to use linear state equations. Conveniently, in the RFC the equations are linear and the transforming matrix is the identity.

$$\begin{bmatrix} \dot{x_C} \\ \dot{y_C} \\ \dot{\omega_C} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\omega_R} \end{bmatrix} \tag{3}$$

Than can be discretized as:

$$_{t+1} = xt + \dot{x}\Delta t \tag{4}$$

The idea is to refer all calculations to the current robot position so the kinematic model is linear, and provide the target coordinates, location of obstacles and all other constraints in the RFC. This way, the non-linear transformation between the robot and the world's frames is computed outside the minimization framework.

3 Model Predictive Control

Model Predictive Control (MPC) is an advanced control method that relies on a model of the dynamics of the system to compute a finite-time horizon. A cost function is optimized over this horizon, resulting in a optimal sequence of control inputs. The first input of this sequence is executed, and the state is read afterwards. Then, all the optimization is performed again. MPC can anticipate future events and can take control actions accordingly.

A general MPC problem can be formulated in discrete time as follows:

$$\begin{gathered}\min_{x,u} \Phi(x,u) = \sum_{t=0}^{N} l(x_t, u_t) \\ s.t. \begin{cases} x_0 = x_S, x_{N+1} = x_F \\ x_{t+1} = f(x_k, uk) \\ g(x_k, u_k) \leq 0 \end{cases}\end{gathered} \tag{5}$$

where N is the prediction horizon, l is the optimization goal or cost function, x_S is the state at the beginning and x_F is the state that has to be reached at the end. f represents the dynamics of the system, that in this case corresponds to the direct kinematics equation derived before. g is the set of inequality constraints that applies to system states and control inputs.

To solve the problem in real-time so the control outputs can be fed to a robot, one of the fastest techniques is to formulate it as a sequence of Quadratic Programming problems, using at each iteration the first solution found. After applying the command to the robot the state is read and a new QP is prepared starting from this new initial state. The approach is shown in Algorithm (1).

Algorithm 1: Sequential Quadratic Programming

```
initialization:
    initial robot position $X^0 = [x_0, \dots, x_N]^0, U^0 = [u_0, \dots, u_N]^0$ ;
    target $T$
while $T$ not achieved do
    1: read state from robot $X^i = [x_0, \dots, x_N]^{i-1}, U^i = [u_0, \dots, u_N]^{i-1}$ ;
    2: prepare a QP problem with new initial state $x_0$ and all constraints in RFC;
    3: solve the QP problem to obtain $u^*$;
    4: send the first control input $u[0]$ to robot;
end
```

3.1 The cost function

The target is to minimize the error between the current state and reference state. The reference control equals zero in to order to force the robot to stop once it reaches the end point. To avoid reaching a negative value when calculating the error, the difference is squared. This also ensures the smoothness of the motion making the correction of the error a quadratic function.

Before discussing about the cost function, **the loss function** must be introduced, since it's considered the main block for the mathematical formulation for the MPC. The loss function is a Linear Quadratic Regulation where the running costs are penalized using L2-norm method. The LQR is given by:

$$l(x,u) = \|x_u - x_r\|_Q^2 + \|u - u_r\|_R^2 \tag{6}$$

Where:

- $x_u \rightarrow$ is the predicted state.
- $x_r \rightarrow$ is the reference state.
- $u \rightarrow$ is the control action.

- u_r $\rightarrow$ is the reference control action.
- Q $\rightarrow$ is the weights for the states.
- R $\rightarrow$ is the weights for the control actions.

In other words, we penalize the squared difference between the predicted state and its reference, added to the squared difference between the control value and its reference. Knowing that Q and R are diagonal matrices for the tuning parameters.

Therefore, **the cost function** is considered the evaluation of the running costs' summation along the whole prediction horizon. It is interpreted by the following formula:

$$J_N(x, u) = \sum_{t=0}^{N} l(x_t, u_t) \tag{7}$$

This shows the dependency between the loss function and the cost function; however, the cost function is considered the more general form since it includes the summation of LQR in each step along the prediction horizon where N is the number of steps in the prediction horizon. This shows the dependency between the loss function and the cost function; however, the last one is considered the more general form since it includes the summation of LQR in each step along the prediction horizon.

Moving to the most general form, **the objective function** which is the main purpose to be reached in MPC. By means, it relies on minimizing the cost function indicating how much each control action contributes to the optimized value in the mathematical optimization problem. This is used for the **Point Stabilization Problem** and can be clearly demonstrated as follows:

$$min_{x,u}\Phi(x, u) = \sum_{t=0}^{N} l(x_t, u_t) \tag{8}$$

3.2 Including obstacles as constraints

3.3 Wheels to robot kinematics

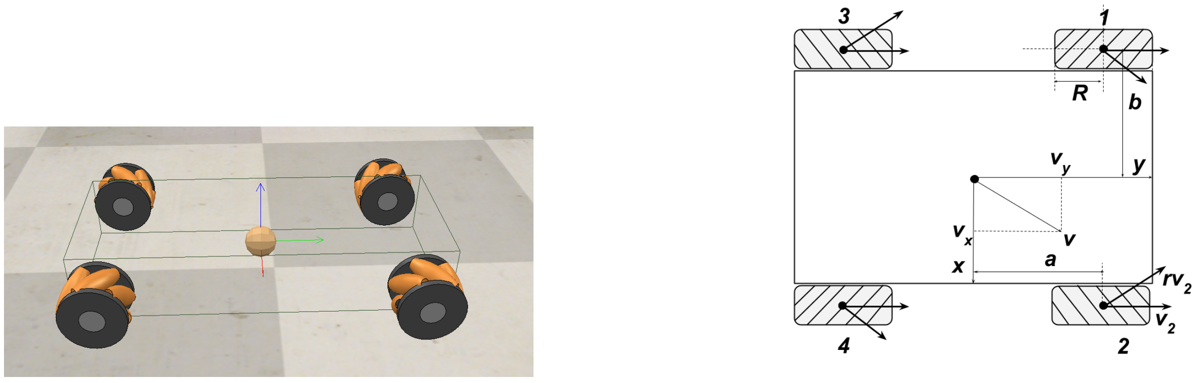

Figure 1: Virtual model of Viriato base.

According to Figure 1, if the velocity vector at the centre C of the robot is v, then the projections of v over the x and y axis of the robot, assuming angle 0 is aligned with the y axis, can be obtained as:

$$\begin{aligned} v_x &= vsin(\theta) \\ v_y &= vcos(\theta) \end{aligned} \tag{9}$$

where θ is the angle that forms the direction of motion with the x of the robot. The angular velocity of the robot ϕ is defined as the turning rate at the centre point C. Each mecanum wheel is defined by four parameters: its coordinates

(a, b) with respect to the robot's coordinate system C, the tilt angle γ of the rollers in each wheel with respect to the robot's y axis, and the radius R of the complete wheel, including the rollers. According to the order of the wheels in 1:

$$\begin{aligned} w_1 &= \{-k_x, k_y, -\frac{\pi}{4}, R\} \\ w_2 &= \{k_x, k_y, \frac{\pi}{4}, R\} \\ w_3 &= \{-k_x, -k_y, \frac{\pi}{4}, R\} \\ w_4 &= \{k_x, -k_y, -\frac{\pi}{4}, R\} \end{aligned} \tag{10}$$

The linear velocity vectors of wheel i and the roller in contact with the ground, are v_i and rv_i respectively.

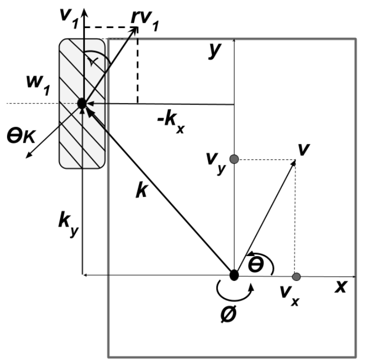

Figure 2: Detail of robot-wheel velocity relationship

From Figure 2 (right), the following pairs of equations relate the linear velocity components of each wheel with the robot's velocity. For wheel 1, as showed in 2, the forward velocity component of the robot is the sum of the v_x projection and the contribution due to current rotation ϕ applied to the k vector also projected on the forward direction. Since a is negative and ϕ rotates counterclockwise,

$$\begin{aligned} v_x - k_x\phi &= rv_1 sin(\gamma_1) \\ v_y + k_y\phi &= v_1 + rv_1 cos(\gamma_1) \end{aligned} \tag{11}$$

Dividing the equation by $tan(\gamma_i)$ and substituting in the first equation gives:

$$v_i = v_y + k_y\phi - \frac{v_x - k_x\phi}{tan(\gamma_i)} \tag{12}$$

Substituting the values of the tangents for $\pi/4$ angles, $tan(\gamma_i) : \{-1, 1, 1, -1\}$:

$$\begin{aligned} v_1 &= v_x + v_y + \phi(k_x + k_y) \\ v_2 &= -v_x + v_y + \phi(k_x + k_y) \\ v_3 &= -v_x + v_y + \phi(-k_x - k_y) \\ v_4 &= v_x + v_y + \phi(-k_x - k_y) \end{aligned} \tag{13}$$

and taking the angular speed of each wheel from the linear velocity equation: $v_i = \omega_i R$, we can write the resulting equations in matrix form as:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \underbrace{\frac{1}{R} \begin{bmatrix} 1 & 1 & k_x + k_y \\ -1 & 1 & k_x + k_y \\ -1 & 1 & -k_x - k_y \\ 1 & 1 & -k_x - k_y \end{bmatrix}}_{I_k} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{14}$$

that represents the inverse kinematics of the robot.

We are interested in building a MPC controller and the control inputs will be the speed of the wheels, so we need the forward kinematics. Since the I_K matrix in 14 is not square, we can solve for the state vector using the pseudo-inverse,

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = I_k^+ \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{15}$$

where $I_k^+ = (I_k^T I_k)^{-1} I_k^T$. We can obtain the forward kinematics matrix D_k symbolically using SimPy.

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = R \underbrace{\begin{bmatrix} \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4(a+b)} & \frac{1}{4(a+b)} & -\frac{1}{4a+4b} & -\frac{1}{4a+4b} \end{bmatrix}}_{F_K} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{16}$$

3.4 Instantation with Viriato's model

We need now to express the direct kinematics equation in 15 as a discrete time linear matrix equation: $x_{t+1} = Ax_k + Buk$. Since the robot can move in any of its three state dimensions $x : \{x, y, \omega\}$ without geometric constraints, we can write:

$$\begin{aligned} x_{t+1} &= xt + f(\omega_1, \omega_2, \omega_3, \omega_4, \delta t) \\ y_{t+1} &= yt + g(\omega_1, \omega_2, \omega_3, \omega_4, \delta t) \\ \phi_{t+1} &= \phi_t + h(\omega_1, \omega_2, \omega_3, \omega_4, \delta t) \end{aligned} \tag{17}$$

where the f, g, h functions map wheel velocities and mechanic parameters into v_x, v_y, w respectively. They come directly from matrix multiplication of DK with the column vector of wheel velocities in 16.

We can express this as a matrix equation as:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \phi_{t+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} x_t \\ y_t \\ \phi_t \end{bmatrix} + R \underbrace{\begin{bmatrix} \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4(a+b)} & \frac{1}{4(a+b)} & -\frac{1}{4a+4b} & -\frac{1}{4a+4b} \end{bmatrix}}_{B} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{18}$$

Pyrep currently uses this inverse kinematics matrix to control the Youbot, all units in $\frac{rads}{sg}$:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & 1 & 1 \end{bmatrix}}_{I_k} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{19}$$

And the real Viriato robot uses this inverse kinematics matrix,

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & -1 & k_x + k_y \\ 1 & 1 & -(k_x + k_y) \\ 1 & 1 & k_x + k_y \\ 1 & -1 & -(k_x + k_y) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{20}$$

Now we need to define the equality constraints

3.5 Differential robot's mode

So far, the subsections discussed the omni-directional drive's perspective. However, we can switch to the differential drive in order to change the robot's direction during motion.

By definition, the differential drive consists of two wheels mounted on a common axis, where each wheel must rotate about a point lying on it know as the Instantaneous Center of Curvature (ICC), and each wheel can be driven independently in a direction. This can be described mathematically as follows:

$$\begin{aligned} v_r &= w(R + l/2) \\ v_l &= w(R - l/2) \end{aligned} \tag{21}$$

Where:

- $v_r \rightarrow$ is the right wheel's velocity.
- $v_l \rightarrow$ is the left wheel's velocity.
- $w \rightarrow$ is the angular velocity of the robot around ICC.
- $R \rightarrow$ is the signed distance from the ICC to the midpoint between the wheels.
- $l \rightarrow$ is the distance between the centers of the two wheels.

By reformulating the equations above, we get:

$$\begin{aligned} R &= \frac{1}{2}\frac{v_r + v_l}{v_r - v_l} \\ w &= \frac{v_r - v_l}{l} \end{aligned} \tag{22}$$

Hence, we can deduce three cases:

- If $v_r = v_l \rightarrow$ it means there's no rotation, R tends to infinity then the robot moves linearly.
- If $v_r = -v_l \rightarrow$ it means there's a rotation around the midpoint, R is equal to zero then the robot rotates in place.
- If $v_r = 0$, or $v_l = 0 \rightarrow$ it means there's a rotation about one side wheel, R is equal to l/2.

In our case, we have a mecanum-wheeled robot where the inverse kinematics model is derived in equation (14), if we consider that we have two wheels instead of four to make the differential's robot design, we will assume that:

$$\begin{aligned} w_1 &= w_3 \\ w_2 &= w_4 \end{aligned} \tag{23}$$

The output vector is:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{R} \begin{bmatrix} v_x + v_y + (k_x + k_y)w \\ -v_x + v_y + (k_x + k_y)w \\ -v_x + v_y - (k_x + k_y)w \\ v_x + v_y - (k_x + k_y)w \end{bmatrix} \tag{24}$$

In order to satisfy the assumed condition, v_x should be equal to zero. This is the idea behind setting v_x to zero in the lower and upper boundaries for the control actions.

3.6 Trajectory Tracking

The trajectory tracking is a time varying reference values of the state vector. According to this definition and comparing it with the point stabilization, the cost function slightly differs, only the reference point will be a function of time instead of a constant value to be reached as a target point. The LQR equation is as follows:

$$l(x, u) = \|x_u - x(t)_r\|_Q^2 + \|u - u(t)_r\|_R^2 \tag{25}$$

By then, the cost function is minimized as:

$$J_N(x, u) = \sum_{k=0}^{N-1} l(x_u(k), u(k)) \tag{26}$$

Where $k \rightarrow$ is the current step in the prediction horizon.

Trajectory tracking is one of the methods that are used to move on a certain path. In the next subsection, a more general approach will be discussed in depth.

3.7 Path Following

The path is defined as a set of connected points that fit a line or a curve. Hence, path following is a control task in which a geometric reference is to be followed without involving time as a constraint. From this basic principle, the path can be formed by a polynomial function with a specific order fitting the points.

We begin with the kinematic model for omni-directional robot:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} cos(x_3) & -sin(x_3) & 0 \\ sin(x_3) & cos(x_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{27}$$

Where,

- $\dot{x_1}$ $\rightarrow$ is the velocity in X-direction.
- $\dot{x_2}$ $\rightarrow$ is the velocity in Y-direction.
- $\dot{x_3}$ $\rightarrow$ is the angular velocity.
- x_3 $\rightarrow$ is the angle in the spacial component.
- u_1 $\rightarrow$ is the velocity in X-direction.
- u_2 $\rightarrow$ is the velocity in Y-direction.
- u_3 $\rightarrow$ is the angular speed.

From the following, we can deduce firstly, the linear velocities:

$$u_1 = \dot{x_1} cos(x_3) + \dot{x_2} sin(x_3) \tag{28}$$

$$u_2 = \dot{x_2} cos(x_3) - \dot{x_1} sin(x_3) \tag{29}$$

And secondly, the angular velocity:

$$u_3 = \dot{x_3} \tag{30}$$

An impressive and a brilliant idea was introduced in "Predictive Path Following of Mobile Robots without Terminal Stabilizing Constraints" paper. It introduces and spots the light on adding a path parameter to the states and treat it as virtual state. This additional state indicates the progress of the robot along the predefined path.

As described earlier, the states are a polynomial function of the path parameter θ as shown below:

$$\begin{aligned} x_1 &= K(\theta) \\ x_2 &= P(\theta) \\ x_3 &= \tan^{-1}(\frac{\partial P}{\partial K}) \end{aligned} \tag{31}$$

Similarly, a virtual control $\dot{\theta}$ is also introduced. It is considered the control action that governs the virtual state's change rate. From substituting equation(31) in (28), (29) and (30); we finally get the control actions as a function of θ and $\dot{\theta}$:

$$u_1 = \dot{\theta} \sqrt{\frac{\partial K}{\partial \theta} + \frac{\partial P}{\partial \theta}} \tag{32}$$

$$u_2 = 0 \tag{33}$$

$$u_3 = \frac{1}{1 + (\frac{\partial P}{\partial K})^2} (\frac{\partial}{\partial t} \frac{\partial P}{\partial K}) \tag{34}$$

From all of the above, the system will be augmented and represented as:

$$\begin{aligned} z &= \begin{bmatrix} x \\ \theta \end{bmatrix} \\ \dot{z} &= \begin{bmatrix} \dot{x} \\ \dot{\theta} \end{bmatrix} \end{aligned} \tag{35}$$

Also, adding constraints for the path parameter is an important step for the optimization as well as tuning the weights. Briefly, the path parameter θ is treated as a state; however, it's only a parameter to show if the current state is on the path and how far the robot is from the end of the path. For the implementation, the open-loop control (u_{ref}) can be disregarded (i.e, x_3 in equation (31), and u_1 and u_3 in equations (28) and (30) respectively). Instead, we can penalize the control action itself to maintain a stable motion.

3.8 CASADI formulation

3.9 Gurobi formulation

3.10 OSQP formulation

3.11 Model Predictive Control with obstacle avoidance

3.12 Model Predictive Control for socially-aware navigation

References

.1 Linearisation

We now want to linearise f with respect to some arbitrary point (x_r, u_r) so we can use the tools of non linear optimization to obtain an optimal trajectory given a set of constraints.

We start by expanding the right side of (2) in Taylor series around the current position of the robot (x_r, u_r). Discarding the high order terms it follows that,

$$f(x + x_r, u + u_r) = f(x_r, u_r) + \frac{\partial f(x,u)}{\partial x}\bigg|_{x=x_, u=u_r} (x - x_r) + \frac{\partial f(x,u)}{\partial u}\bigg|_{x=x_r, u=u_r} (u - u_r) \tag{36}$$

that can also be stated as,

$$f(x + x_r, u + u_r) = f(x_r, u_r) + f_{x_r}(x - x_r) + f_{u_r}(u - u_r) \tag{37}$$

where f_{x_r} and f_{u_r} are the Jacobians of f with respect to x and u respectively, evaluated around the reference point (x_r, u_r). With this equation, points nearby (x_r, u_r) can be evaluated, specifically points in the direction of the target.

To compute these Jacobians we will write first the discrete-time approximation of f using forward differences.

$$x_C^{k+1} = x_C^k + \Delta T * f(x, u) \tag{38}$$

The state position variables now appear in the resulting equation,

$$\begin{bmatrix} x_C^{k+1} \\ y_C^{k+1} \\ \omega_C^{k+1} \end{bmatrix} = \begin{bmatrix} x_C^k \\ y_C^k \\ \omega_C^k \end{bmatrix} + \begin{bmatrix} cos(\omega_C^k) & -sin(\omega_C^k) & 0 \\ sin(\omega_C^k) & cons(\omega_C^k) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_w \end{bmatrix} \tag{39}$$

where

$$\begin{aligned} u_x &\approx \dot{x}_R \Delta T \\ u_y &\approx \dot{y}_R \Delta T \\ u_w &\approx \dot{w}_R \Delta T \end{aligned} \tag{40}$$

Now, to facilitate the derivation we can write the row equations explicitly,

$$\begin{aligned} x_C^{k+1} &= x_C^k + u_x cos(\omega_C^k) - u_y sin(\omega_C^k)) \\ y_C^{k+1} &= y_C^k + u_x sin(\omega_C^k) + u_y cos(\omega_C^k)) \\ \omega_C^{k+1} &= \omega_C^k + u_w \end{aligned} \tag{41}$$

and take the derivatives of each equation with respect to the state variables (x, y, ω) to form matrix A with dimension (SxS),

$$A = \begin{bmatrix} \frac{\partial x_C^{k+1}}{\partial x_C^k} = 1 & \frac{\partial x_C^{k+1}}{\partial y_C^k} = 0 & \frac{\partial x_C^{k+1}}{\partial \omega_C^k} = -u_x sin(\omega_C^k) - u_y cos(\omega_C^k) \\ \frac{\partial y_C^{k+1}}{\partial x_C^k} = 0 & \frac{\partial y_C^{k+1}}{\partial y_C^k} = 1 & \frac{\partial y_C^{k+1}}{\partial \omega_C^k} = u_x cos(\omega_C^k) - u_y sin(\omega_C^k)) \\ \frac{\partial \omega_C^{k+1}}{\partial x_C^k} = 0 & \frac{\partial \omega_C^{k+1}}{\partial y_C^k} = 0 & \frac{\partial \omega_C^{k+1}}{\partial \omega_C^k} = 1 \end{bmatrix} \tag{42}$$

and to the input variables (u_x, u_y, u_w) to obtain matrix B with dimension (SxC),

$$B = \begin{bmatrix} \frac{\partial x_C^{k+1}}{\partial \dot{x}_R} = cos(\omega_C^k) & \frac{\partial x_C^{k+1}}{\partial \dot{x}_R} = -sin(\omega_C^k) & \frac{\partial x_C^{k+1}}{\partial \dot{\omega}_R} = 0 \\ \frac{\partial y_C^{k+1}}{\partial \dot{x}_R} = sin(\omega_C^k) & \frac{\partial y_C^{k+1}}{\partial \dot{y}_R} = cos(\omega_C^k) & \frac{\partial y_C^{k+1}}{\partial \dot{\omega}_R} = 0 \\ \frac{\partial \omega_C^{k+1}}{\partial \dot{x}_R} = 0 & \frac{\partial \omega_C^{k+1}}{\partial \dot{x}_R} = 0 & \frac{\partial \omega_C^{k+1}}{\partial \dot{\omega}_R} = \Delta T \end{bmatrix} \tag{43}$$

We can now build the linear equation that we need for the optimization algorithm,

$$x_C^{k+1} = A(x_C^k - x_r) + B(u_k - u_r) \tag{44}$$

This equation defines a tangent plane to f at (w_C, u_x, u_y). We can now move on it towards the target using linear tools but the approximation is only good in the vicinity of the point.

.2 Rearrange as a QP program

For each inner step of this iterative process we have to write the objective function Φ to be minimized. The simplest form is the sum of two terms, one representing the sum of the magnitudes of the state vectors, and a second one representing the sum of the magnitudes of the control inputs.

$$\Phi(k) = \sum_{j=1}^{N} (x_{k+j|k})^T Q (x_{k+j|k})^T + u_{k+j-1|k}^T R u_{k+j-1|k} \tag{45}$$

where N is the prediction horizon and $Q \geq 0$, $R > 0$ are weighting matrices. The notation $a(m|n)$ indicates the value a will have at instant m, predicted from the value at instant n.

The optimization problem reduces now to find u^* such that,

$$\tilde{u}^* = \min_{\tilde{u}} \Phi(k) \tag{46}$$

We need to solve two problems before we can proceed:

1. we need to build a stacked form for the linearised model equation, where all time steps are packed into the model matrices,
2. and afterwards, we need to rewrite it as a standard quadratic programming problem:

$$\begin{aligned} \min_{w} Q(w) &= \frac{1}{2} w^t H w + f^T w \\ s.t. \qquad & w_{min} \leq Aw \leq w_{max} \end{aligned} \tag{47}$$

.3 Stacking time into space

Let's start with the problem of stacking the time expansion required by the MPC, into a vector or states.

We start by defining the following vectors:

$$\bar{x}(k+1) = \begin{bmatrix} \tilde{x}(k+1|k) \\ \tilde{x}(k+2|k) \\ \vdots \\ \tilde{x}(k+N|k) \end{bmatrix} \qquad \bar{u}(k) = \begin{bmatrix} u(k+1|k) \\ u(k+2|k) \\ \vdots \\ u(k+N|k) \end{bmatrix} \tag{48}$$

where the index $k : \{1 : N\}$ stands for the N prediction steps that the MPC will explore in each iteration.

With these new variables, the cost function can be rewritten as:

$$\Phi(k) = \bar{x}^T(k+1)\bar{Q}\bar{x}(k+1) + \bar{u}(k)^T\bar{R}\bar{u}(k) \tag{49}$$

with

$$\bar{Q} = \begin{bmatrix} Q & 0 & \dots & 0 \\ 0 & Q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Q \end{bmatrix} \qquad \bar{R} = \begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R \end{bmatrix} \tag{50}$$

From here it is possible to write the linearised model equation for the complete step as:

$$\bar{x}(k+1) = \bar{A}(k)\tilde{x}(k|k) + \bar{B}\bar{u}(k) \tag{51}$$

Note that $\tilde{x}(k|k)$ is the state at the beginning of the step, i.e. the same value in all vector elements, while $\bar{u}(k)$ is the vector of all the initial control inputs for each step of the k sequence. The details will be clearer as we proceed.

The matrices $\bar{A}$ and $\bar{B}$ must encode the sequence of model transformations from step k to step $k+N$. Starting from k, each successive position of the robot is achieved by applying a new control input u to the model equation. We can start by writing the equations defining each step transition:

$$\begin{aligned} &\tilde{x}(k+1|k) = A(k)\tilde{x}(k|k) + B(k)\tilde{u}(k) \\ &\tilde{x}(k+2|k) = A(k+1)\tilde{x}(k+1|k) + B(k+1)\tilde{u}(k+1) \\ &\dots \\ &\tilde{x}(k+n|k) = A(k+n)\tilde{x}(k+1|k) + B(k+1)\tilde{u}(k+n) \end{aligned} \tag{52}$$

We can now rewrite this set of equations unfolding the recurring relations and grouping again the vector of all $\tilde{x}(k+i|k)$ as $\bar{x}(k+1)$

$$\bar{x}(k+1) = \begin{bmatrix} A(k)\tilde{x}(k|k) + B(k)\tilde{u}(k) \\ A(k+1)(A(k)\tilde{x}(k|k) + B(k)\tilde{u}(k)) + B(k+1)\tilde{u}(k+1) \\ A(k+2)(A(k+1)(A(k)\tilde{x}(k|k) + B(k)\tilde{u}(k)) + B(k+1)\tilde{u}(k+1)) + B(k+2)\tilde{u}(k+2) \\ \dots \end{bmatrix} \tag{53}$$

and by moving out $\tilde{x}(k|k)$ and $\tilde{u}(k+i)$, we can now build $\bar{A}$,

$$A = \begin{bmatrix} A(k) & 0 & 0 & \dots & 0 \\ 0 & A(k+1)A(k) & 0 & \dots & 0 \\ 0 & 0 & A(k+2)A(k+1)A(k) & \dots & 0 \\ \dots & & & & \\ & & & \prod_{i=0}^{N} A(k+i) & \end{bmatrix} \tag{54}$$

Matrix A has dimension $(SN)x(SN)$ where S is the dimension of the state vector (3 in this case).

The matrix $\bar{B}$ is:

$$B = \begin{bmatrix} B(k) & 0 & 0 & 0 & \dots & 0 \\ A(k+1)B(k) & B(k+1) & 0 & 0 & \dots & 0 \\ A(k+2)A(k+1)B(k) & A(k+2)B(k+1) & B(k+2) & 0 & \dots & 0 \\ A(k+3)A(k+2)A(k+1)B(k) & A(k+3)A(k+2)B(k+1) & A(k+3)B(k+2) & B(k+2) & \dots & 0 \\ & \dots & & & & \end{bmatrix} \tag{55}$$

with dimension $(SNxCN)$.

.4 Reformulation as standard QP problem

Now that we have the time-stacked linearised model equation:

$$\bar{x}(k+1) = \bar{A}(k)\tilde{x}(k|k) + \bar{B}\bar{u}(k) \tag{56}$$

we use it to substitute $\bar{x}(k+1)$ in (49):

$$\Phi(k) = (\bar{A}(k)\tilde{x}(k|k) + \bar{B}\bar{u}(k))^T Q\bar{A}(k)\tilde{x}(k|k) + \bar{B}\bar{u}(k) + \bar{u}(k)^T\bar{R}\bar{u}(k) \tag{57}$$

and now group terms to rewrite $\Phi(k)$ in standard quadratic form:

$$\Phi(k) = \frac{1}{2}\bar{u}^T(k)H(k)\bar{u}(k) + f^T(k)\bar{u}(k) + d(k) \tag{58}$$

The first term gathers the quadratic terms in the control variable u. The second term is linear in u and groups the state variables x. The third term does not depend on u and can be dropped from the minimization. H is the Hessian of the cost function and f is the gradient, although I still don't know why.

Expanding (59):

$$\begin{aligned} \Phi(k) = {} & (\bar{A}(k)\tilde{x}(k|k))^T Q\bar{A}(k)\tilde{x}(k|k) + (\bar{A}(k)\tilde{x}(k|k))^T Q\bar{B}\bar{u}(k) + (\bar{B}\bar{u}(k))^T Q\bar{A}(k)\tilde{x}(k|k) + \\ & (\bar{B}\bar{u}(k))^T Q\bar{B}\bar{u}(k) + \bar{u}(k)^T\bar{R}\bar{u}(k) \end{aligned} \tag{59}$$

and grouping quadratic, linear and independent terms in u we get expressions for the three terms of the cost function:

$$\begin{aligned} H(k) &= 2(\bar{B}^T(k)\bar{Q}\bar{B}(k) + \bar{R}) \\ f(x) &= 2\bar{B}^T(k)\bar{Q}\bar{A}(k)\tilde{x}(k|k) \\ d(k) &= \tilde{x}^T(k|k)\bar{A}^T(k)\bar{Q}\bar{A}(k)\tilde{x}(k|k) \end{aligned} \tag{60}$$

This expression, along with the constraints on the state and control inputs, can be passed to the QP solver in each iteration of the SQP loop described in (1).