
Real-Time Control of a Mobile Robot Using Linearized Model Predictive Control

Table of Contents

Problem Formulation	1
Proposed Solution	2
MPC	2
Constraints	4
Results	67

Olzhas Adiyatov (olzhas.adiyatov@uwaterloo.ca) and Yifeng Wang (y3438wang@uwaterloo.ca)

Problem Formulation

Control of the mobile robot is an essential problem in robotics. In this project we will address the problem of trajectory following by a mobile robot with a unicycle kinematic mobile robot model. The figure below depicts the unicycle kinematic mobile robot model.

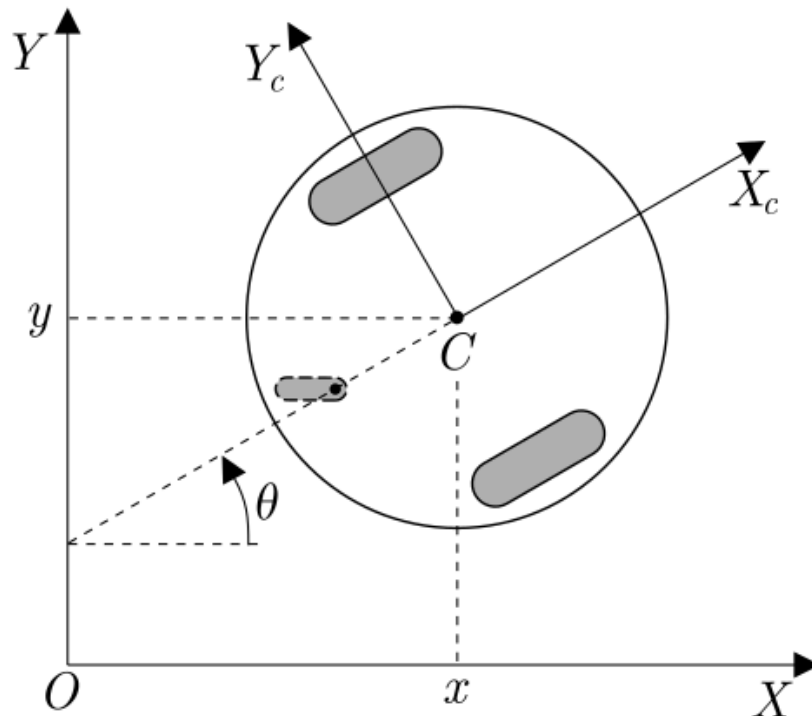


Figure 1. Unicycle kinematic mobile robot model

let $\mathbf{x} := [x, y, \theta]^T$ be a configuration state of the robot, Cartesian coordinates and orientation center of wheels C w.r.t. a global inertial frame (O, X, Y) , and $\mathbf{u} := [v, w]^T$ be a control input vector.

The mobile robot is governed by the following equation :

$$\dot{x} = v \cos \theta$$

$$\dot{y} = v \sin \theta$$

$$\dot{\theta} = w$$

the above system model in a compact form is:

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

The system above is nonlinear, i.e. we cannot easily use LQT to track the reference signal. Let $\mathbf{x}_r, \mathbf{u}_r$ be a reference point, hence the model at those values will be:

$$\dot{\mathbf{x}}_r = f(\mathbf{x}_r, \mathbf{u}_r)$$

Now using Taylor series around point $(\mathbf{x}_r, \mathbf{u}_r)$

$$\dot{\mathbf{x}} = f(\mathbf{x}_r, \mathbf{u}_r) + \left. \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \right|_{(\mathbf{x}_r, \mathbf{u}_r)} (\mathbf{x} - \mathbf{x}_r) + \left. \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \right|_{(\mathbf{x}_r, \mathbf{u}_r)} (\mathbf{u} - \mathbf{u}_r)$$

or

$$\dot{\mathbf{x}} = f(\mathbf{x}_r, \mathbf{u}_r) + f_{\mathbf{x}_r}(\mathbf{x} - \mathbf{x}_r) + f_{\mathbf{u}_r}(\mathbf{u} - \mathbf{u}_r)$$

where $f_{\mathbf{x}_r}$ and $f_{\mathbf{u}_r}$ are the Jacobians of f w.r.t. \mathbf{x} and \mathbf{u} around a reference point $(\mathbf{x}_r, \mathbf{u}_r)$. Now, let us define $\tilde{\mathbf{x}} := \mathbf{x} - \mathbf{x}_r$ and $\tilde{\mathbf{u}} := \mathbf{u} - \mathbf{u}_r$, hence the system model can be reformulated as:

$$\dot{\tilde{\mathbf{x}}} = f_{\mathbf{x}_r} \tilde{\mathbf{x}} + f_{\mathbf{u}_r} \tilde{\mathbf{u}}$$

Now, we can discretize the system model with forward differences with sampling time T and k is a sampling interval:

$$\tilde{\mathbf{x}}(k+1) = \mathbf{A}(k)\tilde{\mathbf{x}}(k) + \mathbf{B}\tilde{\mathbf{u}}(k)$$

with

$$\mathbf{A}(k) := \begin{bmatrix} 1 & 0 & -v_r(k) \sin \theta_r(k)T \\ 0 & 1 & v_r(k) \cos \theta_r(k)T \\ 0 & 0 & 1 \end{bmatrix} \mathbf{B}(k) := \begin{bmatrix} \cos \theta_r(k)T & 0 \\ \sin \theta_r(k)T & 0 \\ 0 & T \end{bmatrix}$$

Proposed Solution

MPC

Model Predictive Control (MPC) is a optimization-based advanced control technique. MPC requires a system model of the controlled robotic system. Linear MPC is good. Often times a model of a control system is non-linear linearization might be a solution to this problem.

$$\Phi(k) = \sum_{j=1}^N \tilde{\mathbf{x}}^T(k+j|k) \mathbf{Q} \tilde{\mathbf{x}}(k+j|k) + \tilde{\mathbf{u}}^T(k+j-1|k) \mathbf{R} \tilde{\mathbf{u}}(k+j-1|k)$$

where N is a horizon length and $\mathbf{Q} \geq 0$ (positive semidefinite), $\mathbf{R} > 0$ (positive definite).

The optimization problem can be stated as

$$\tilde{\mathbf{u}}^* = \arg \min_{\tilde{\mathbf{u}}} \{ \Phi(k) \}$$

The above formulated optimization problem is solved at each time iteration k . Eventhough the result of the optimization is a sequence of control inputs we apply only the first $\tilde{\mathbf{u}}$.

Let's reformulate the MPC problem as a regular constrained QP problem. To transform the problem into QP we introduce helpers vectors:

$$\bar{\mathbf{x}}(k+1) := \begin{bmatrix} \tilde{\mathbf{x}}(k+1) \\ \tilde{\mathbf{x}}(k+2) \\ \vdots \\ \tilde{\mathbf{x}}(k+N) \end{bmatrix} \quad \bar{\mathbf{u}}(k) := \begin{bmatrix} \tilde{\mathbf{u}}(k) \\ \tilde{\mathbf{u}}(k+1) \\ \vdots \\ \tilde{\mathbf{u}}(k+N-1) \end{bmatrix}$$

Now cost function $\Phi(k)$ can be rewritten in the following form:

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

$$\bar{\mathbf{Q}} := \begin{bmatrix} \mathbf{Q} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{Q} \end{bmatrix} \quad \bar{\mathbf{R}} := \begin{bmatrix} \mathbf{R} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{R} \end{bmatrix}$$

where

Using vector $\bar{\mathbf{x}}(k+1)$ it also becomes possible to write the state equation as the following equation:

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

with

$$\bar{\mathbf{A}}(k) := \begin{bmatrix} \alpha(k+1, k) \\ \alpha(k+2, k) \\ \vdots \\ \alpha(k+N, k) \end{bmatrix}$$

and

$$\bar{\mathbf{B}}(k) := \begin{bmatrix} \beta_{11}(k) & \mathbf{0} & \dots & \mathbf{0} \\ \beta_{21}(k) & \beta_{22}(k) & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{N1}(k) & \beta_{N2}(k) & \dots & \beta_{NN}(k) \end{bmatrix}$$

where $\alpha(k_1, k_0)$ and β_{ij} are defined as:

$$\alpha(k_1, k_0) := \begin{cases} \mathbf{I} & \text{if } k_1 = k_0 \\ \prod_{i=0}^{n-1} \mathbf{A}(k+i) & \text{if } k_1 > k_0 \end{cases}$$

$$\beta_{ij}(k) := \alpha(k+i, k+j)\mathbf{B}(k+j-1)$$

Finally, plugging in the state transformation matrices we can write the objective function in standard quadratic form:

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

with

$$\mathbf{H}(k) := 2(\bar{\mathbf{B}}^T(k)\bar{\mathbf{Q}}\bar{\mathbf{B}}(k) + \bar{\mathbf{R}})$$

$$\mathbf{f}(k) := 2\bar{\mathbf{B}}^T(k)\bar{\mathbf{Q}}\bar{\mathbf{A}}(k)\tilde{\mathbf{x}}(k|k)$$

$$\mathbf{d}(k) := \tilde{\mathbf{x}}^T(k|k)\bar{\mathbf{A}}^T(k)\bar{\mathbf{Q}}\bar{\mathbf{A}}(k)\tilde{\mathbf{x}}(k|k)$$

$\mathbf{d}(k)$ is not important term, because it is just a constant that does not affect the objective function.

Constraints

The system might have limitations on control inputs of the system due to its physical limitations, for example, in real world scenario you cannot supply infinite amount of current to an electrical motor or infinite amount of torque to a robot arm or wheel.

$$\mathbf{u}_{min} \leq \mathbf{u}(k) \leq \mathbf{u}_{max}$$

Sometimes you also would like to have constraints on your states or outputs, we can account for that using state evolution matrix. MATLAB's quadprog() routine accepts the following form of QPs:

Solver for quadratic objective functions with linear constraints.

quadprog finds a minimum for a problem specified by

$$\min_x \frac{1}{2} x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \leq b, \\ Aeq \cdot x = beq, \\ lb \leq x \leq ub. \end{cases}$$

H , A , and Aeq are matrices, and f , b , beq , lb , ub , and x are vectors.

Since, the decision variable in the formulated problem is $\bar{\mathbf{u}}$, it would be rather straightforward to account for these constraints. we need to formulate the constraint in " $Ax \leq b$ " form. In this case, since $\bar{\mathbf{u}}$ consists of $\tilde{\mathbf{u}}$ we will use the latter variable for brevity.

$$A\tilde{\mathbf{u}} \leq b$$

which becomes

$$\begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} \tilde{\mathbf{u}} \leq \begin{bmatrix} \tilde{\mathbf{u}}_{max} \\ -\tilde{\mathbf{u}}_{min} \end{bmatrix}$$

recalling that $\tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u}_r$ we can finally write that

$$\begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} \tilde{\mathbf{u}} \leq \begin{bmatrix} \mathbf{u}_{max} - \mathbf{u}_r \\ \mathbf{u}_r - \mathbf{u}_{min} \end{bmatrix}$$

```

clf;
clear;

% path specifications

T = pi*50;
dt = 0.05;
K = ceil(T/dt)+50;
v_robot = 0.09;

x_path = 0*ones(1,10);
x_path = [x_path cumsum(v_robot*dt*ones(1,K))];

y_path = (sin(x_path));
% x_path = 1.25*(cos(x_path+pi)+1);

theta_path = zeros(size(x_path));

v_input = zeros(size(1000,1));
w_input = zeros(size(1000,1));

%prediction specifications
np = 6; %the prediction window
nx = 3; %the number of states
nu = 2; %the number of inputs

X_log = zeros(nx,K);

%setting up the specification of the differential drive robot
robotCurrentLocation = [0 0.25];
robotGoal = [x_path(end) y_path(end)];
initialOrientation = pi/2;
robotCurrentPose = [robotCurrentLocation initialOrientation];

syms v_d phi_d T_sample
Ad = [1 0 -v_d*sin(phi_d)*T_sample; 0 1 v_d*cos(phi_d)*T_sample; 0 0 1];
Bd = [cos(phi_d)*T_sample 0; sin(phi_d)*T_sample 0; 0 T_sample];
Cd = [1 0 0; 0 1 0; 0 0 1];
%in the paper it only requires Q to be positive semi definite matrix
and
%R>0 as the weighting matrices for both states and inputs
Q = diag(repmat([1e3 1e3 1], 1, np));

```

Real-Time Control of a Mobile Robot Using Linearized Model Predictive Control

```
R = diag(repmat([1 1], 1, np));

%please note that the paper did not explicitly say anything about
%having a state constraints, so we assume there is no state
%constraints associated

%input constraints
vel_max = 0.2;
vel_min = 0;
w_max = pi/6;
w_min = -pi/6;

%formulate input constraints into matrix format for quadratic
programming
%solver to solve for the best control options for the MPC controller
[u_max, u_min] =
    mpc_follow_functions.formulate_input_con(vel_max,vel_min,w_max,w_min,np,nu);

Ts = 0.05; %the sampling time

%visualize the sinusoidal path
plot(x_path,y_path,'r--')
hold on
xlim([-5 15]);
ylim([-3 3]);
grid on;

goalRadius = 0.2;
distanceToGoal = norm(robotCurrentLocation - robotGoal);

%plot the initial position of the robot
[vx, vy] = mpc_follow_functions.pmark(robotCurrentPose(1),
    robotCurrentPose(2), robotCurrentPose(3), 0.05);
fill(vx, vy, [0.25,0.41,0.88])
% axis square
hold on
drawnow

k=1;

options = optimoptions('quadprog','Display','off');

time_rec = zeros(1,K);
tic;
while(k<K)
    x_robot = robotCurrentPose(1,1);
    y_robot = robotCurrentPose(1,2);
    phi_robot = robotCurrentPose(1,3);
    x_state = [x_robot; y_robot; phi_robot];
    %
    % the below function is to set up the references in matrix for the
    state
    % and the velocity reference used in the model after linearization
    at
```

Real-Time Control of a Mobile Robot Using Linearized Model Predictive Control

```

% the reference point and after discretization and the below
function is
% made for obtain reference of a sinusoidal path. Also note that
the
% paper does not explain how to obtain the linear reference
velocity for
% calculaing each Ad matrix and Bd matrix at each time instance

    [r_val,r_phi,v_ref] =
mpc_follow_functions.obtain_reference(x_path,y_path,theta_path,x_robot,y_robot,np)
    plot(r_val(1:3:end),r_val(2:3:end),'.');
% v_ref = 0*v_ref;

% based on quadratic programming and this part calculates the
quadratic
% objective term, H and linear quadratic term, f
    [H,f] =
mpc_follow_functions.grad_hess(Q,R,x_state,Ad,Bd,Cd,r_phi,r_val,v_ref,np,nx,nu,Ts)

% use the matlab built in quadratic programming to find the
optimal
% contol with H as the Hessian matrix and the vector f to describe
the
% linear part. The standard quadratic form is given as min
% 1/2*u'H*u+f'u+d. Since d is independent of u and does not
matter for
% the determination of u, therefore d term can be excluded.
% H = (H+H')/2;
    [u_input,fval,exitflag] = quadprog(H,f,[],[],[],[],u_min,u_max,
[],options);
    v = u_input(1);
    w = u_input(2);

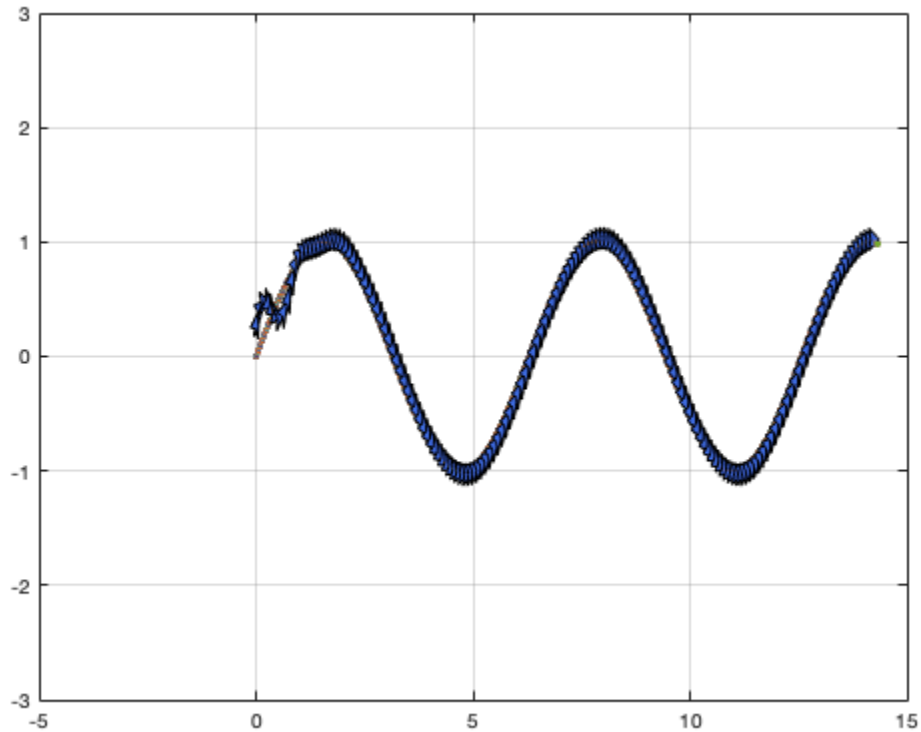
    x_robot = x_robot + v*cos(phi_robot)*Ts;
    y_robot = y_robot + v*sin(phi_robot)*Ts;
    phi_robot = phi_robot + w*Ts;
    time_rec(k)=toc;
    if mod(k,20) == 0
        [vx, vy] = mpc_follow_functions.pmark(x_robot, y_robot,
phi_robot, 0.1);
        fill(vx, vy, [0.25,0.41,0.88])
% axis square
        hold on
        drawnow
    end

% store the current robot position
    robotCurrentPose = [x_robot y_robot phi_robot];
    X_log(:,k)=robotCurrentPose;
% Re-compute the distance to the goal
    distanceToGoal = norm(robotCurrentPose(1:2) - robotGoal);
    k=k+1;

end

```


Warning: Your Hessian is not symmetric. Resetting $H=(H+H')/2$.
Warning: Your Hessian is not symmetric. Resetting $H=(H+H')/2$.
Warning: Your Hessian is not symmetric. Resetting $H=(H+H')/2$.
Warning: Your Hessian is not symmetric. Resetting $H=(H+H')/2$.
Warning: Your Hessian is not symmetric. Resetting $H=(H+H')/2$.



Results

The mobile robot follows the assigned reference path. At the beginning it is possible to notice that there is some kind of overshoot, until the robot hit the reference signal. We noticed that if the mobile robot has an initial state that is sufficiently far away from the path the controller will not be able to follow the trajectory. We think that the possible reason that linearization around the reference points does not provide an accurate model which can be used to find an optimal control action, however if the robot starts on a trajectory or really close to the trajectory we will have a good tracking performance.

```
figure;  
  
dt_results = diff(time_rec(1:end-1));  
  
hist(dt_results,50);  
title('Histogram of timings for our MPC implementation');  
xlabel('Time (s)');  
ylabel('Iterations');  
  
% average time of iteration is  
mean(dt_results)
```

Real-Time Control of a Mobile Robot Using Linearized Model Predictive Control

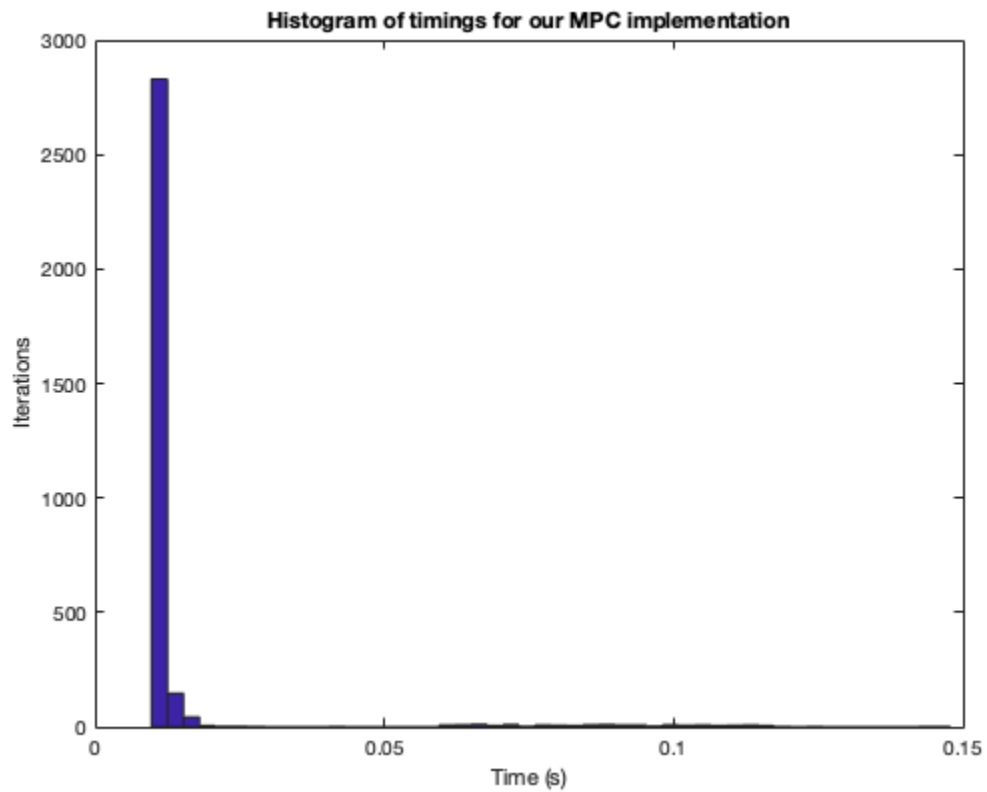
```
% The sampling time for the control algorithm is 0.05s, which means we  
would be able  
% to control the mobile robot.  
% Let us compute the tracking error for the simulation experiment.  
mean_squared_error = sqrt(sum(sum([x_path(1:K); y_path(1:K)] -  
X_log(1:2,:)).^2)))
```

ans =

0.0151

mean_squared_error =

15.0140



Published with MATLAB® R2018a