Developing an MPC Controller for the Actual Car-Type Robot to Operate in 2D Environment while Avoiding Objects

Student: Behnam Moradi

Supervisor: Professor M. Mehrandezh, Faculty of Eng. and Applied Science, URegina

Abstract. The overall aim of this project, was to develop an MPC controller for Donkey Car robot to avoid objects while it is operating inside a specific 2D environment. A Donkey Car robot was provided with Raspberry Pi 4 control unit running Raspbian Operating System. On the other hand, MATLAB support package for R-Pi was used to create connection between MATLAB and R-Pi. CasADi library was used in order to implement MPC controller in MATLAB with pre-defined constraints on control actions and also pre-defined objects inside the operation environment. Finally, a comprehensive algorithm was generated to control the actual robot from MATLAB simulation while avoiding constraints.

Introduction. I this section, the whole process for this project will be discussed. At the very first step, the optimization process is discussed using casADi library. Several simple example were solved using the mentioned library in MATLAB. At the next step, the dynamic model of the robot was extracted and a constrained MPC controller was designed for the robot. At the third step, an object avoidance algorithm was implemented within the MPC controller. And finally, at the last step, a trajectory tracking algorithm was implemented within the MPC controller.

1 The CasADi Library and its Optimization Approach

Nowadays, optimization algorithms are used in many applications from diverse areas, from business to science and engineering. In terms of business, optimization algorithms help us to allocate resources in logistics and investments. When it comes to natural sciences and engineering, it helps us to estimate and fit the model to the measurement data, design and operate the technical and intelligent systems such as car type robots, aircrafts, digital devices, etc.

The first question one might has in mind is that what characterizes an optimization problem. An optimization problem consists of the following three ingredients:

- An objective function, **phi(w)**, which is going to be minimized or maximized. We are trying to minimize the difference between the output of the model and the training data.
- Decision variables, w, that can be chosen. This is a variable that we would like to manipulate it within our optimization algorithm.
- Constraints that shall be respected. Both the equality constraints, g(w) = 0, and inequality constraints, g(w)>=0.

1.1 Mathematical Formulation

Within this project, the non-linear programming problem (NLP) is considered to be our basic concept of standard mathematical in the numerical optimization. Thus, this is how we write down our optimization problem. We try to minimize the objective while we respect the equality and inequality constraints.

$$\begin{cases} \min\limits_{w} \phi(w) \\ g_1(w) \le 0 \\ g_2(w) = 0 \end{cases}$$ and the constraints are assumed to be differentiable.

So basically, we are looking for a W which minimizes the objective function.

Within this project, the CasADi library is used to solve our optimization problem. Generally speaking, CasADi has a comprehensive scope of the numerical optimization. In particular, it facilitates the solution of non-linear programming. Technically, there are 4 standard problem that can be solved and handled by CasADi:

- Quadratic Programming
- Non-Linear Programming
- Root Finding Problems
- Initial-Value Problems in ODE/DAE

Installation of this this library is strait forward. The source code can be downloaded from its website and we just need to change the MATLAB path to its directory. Then we will be able to use the functions.

1.2 Some Examples with CasADi

Within this subsection, 2 examples are introduced that are going to be solved using CasADi library.

1.2.1 Example 1

$\phi(w) = w^2 - 6w + 13$, we need to find the proper "w" which minimizes this objective function.

Step 1:

Navigate MATLAB to the CasADi directory and then Start a .m script.

Start the .m script by the following lines:

```
clc; clear all; close all;
import casadi.*
x = SX.sym('w'); % the decision variable
obj = x^2-6*x+13; % the objective function
g = []; %Optimization Constraints
P = []; % Optimization Problem Parameters

OPT_variables = x;
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);
% Optimizer Parameter - Interior Point Optimization
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0;
opts.print_time = 0;
opts.ipopt.acceptable_tol = 1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;
% Defining a new object which is our solver for the optimization problem
solver = nlpsol('solver', 'ipopt', nlp_prob, opts);

% Constraints
args = struct;
args.lbx = -inf; % Lower bound of the decision variable
args.ubx = inf; % Upper bound of decision variable
args.lbg = -inf; % Lower bound of constraints for the vector g which
consists of other constraints
args.ubg = inf; % Upper bound of constraints for the vector g which
consists of other constraints
args.p = []; % Optimization Parameters
args.x0 = -0.5; % Initialization of the optimization problem

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg, 'p', args.p);
x_sol = full(sol.x); % the desired solution or minimizer
min_value = full(sol.f); % The value of the function at the minimizer point
```

1.2.2 Example 2:

For the following set of data points, fit a straight line of the form: $y = m.x + c$

Within this example, we minimize the sum of the squared errors, between the line and the data points. This method is called minimum least squares.

We have got two optimization variables.

The objective function is simply the sum of the squared error.

$$\underset{m,c}{\phi(m,c)} = \sum_{i=1}^{n_{data}} (y(i) - (m.x(i) - c))^2$$

```
clc; clear all; close all
import casadi.*

% The data points visualization
x = [0 45 90 135 180];
y = [667 661 757 871 1210];

line_width = 2;
fontaize_labels = 16;

set(0, 'DefaultAxesFontName', 'Camberia');
set(0, 'DefaultAxesFontSize', fontaize_labels);

figure(1);

plot(x, y, '*b', 'Linewidth', line_width);
xlabel('x');
ylabel('y');
grid on
box on
hold on

% Optimization Variables
m = SX.sym('m');
c = SX.sym('c');

% Optimization Function
obj = 0;
for i = 1:length(x)
   obj = obj + (y(i) - (m*x(i)+c))^2;
end
```

```
% Optimizer Parameter - Interior Point Optimization
opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level = 0;
opts.print_time = 0;
opts.ipopt.acceptable_tol = 1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

% Defining a new object which is our solver for the optimization problem
solver = nlpsol('solver', 'ipopt', nlp_prob, opts);

% Constraints
args = struct;
args.lbx = -inf; % Lower bound of the decision variable
args.ubx = inf; % Upper bound of decision variable
args.lbg = -inf; % Lower bound of constraints for the vector g which
consists of other constraints
args.ubg = inf; % Upper bound of constraints for the vector g which
consists of other constraints

args.p = []; % Optimization Parameters
args.x0 = [0.5, 1]; % Initialization of the optimization problem

sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
    'lbg', args.lbg, 'ubg', args.ubg, 'p', args.p);
x_sol = full(sol.x);
min_value = full(sol.f);

x_line = [0:1:180];
m_sol = x_sol(1);
c_sol = x_sol(2);
y_line = m_sol*x_line+c_sol;
figure(1);
plot(x_line, y_line, '-k', 'LineWidth', line_width);
legend('Data Points', 'y = 2.88x+574');
```

2 Model Predictive Control Implementation – No object avoidance algorithm is considered in this step

Within this section, a simple kinematic model was extracted for the mobile robot which is called Donkey Car. Then, using CasADi library, an MPC controller was developed to control both the simulated robot and the actual robot which starts moving from point A and stops at point B.

2.1 Kinematic Model of the Robot

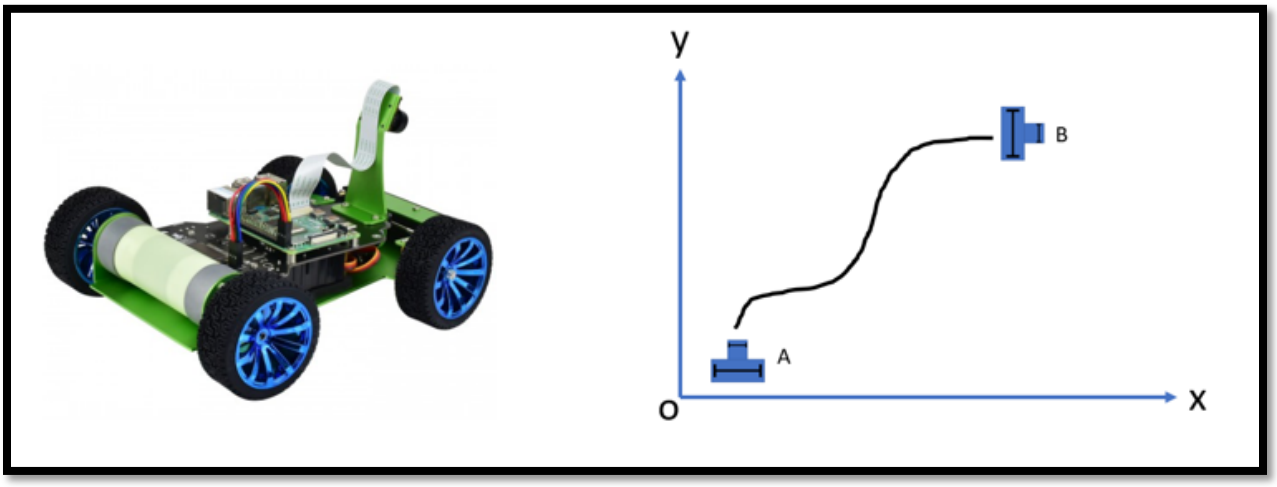


Figure 1. Donkey Car Robot – Powered by Raspberry Pi

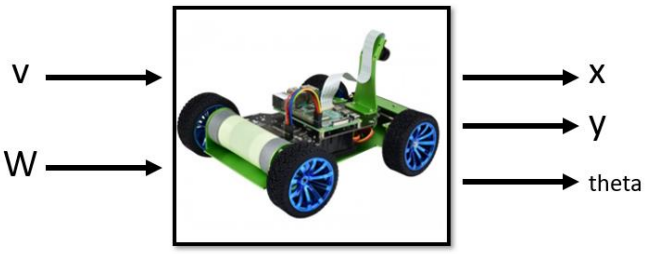
Inputs of the Robot:

1- Linear Velocity (v)
2- Angular Velocity (w)

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

States of the robot:

1- X Position (x)
2- Y Position (y)
3- Angular Position (Theta)
4-



2.2 MPC Formulation

The discrete model of the system was extracted:

$$\dot{\mathbf{x}}(t)=\mathbf{f}_c(\mathbf{x}(t),\mathbf{u}(t)) \qquad \mathbf{x}(k+1)=\mathbf{f}(\mathbf{x}(k),\mathbf{u}(k))$$

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix}v\cos\theta\\ v\sin\theta\\ \omega\end{bmatrix} \xrightarrow[\text{Sampling Time } (\Delta T)]{\text{Euler Discretization}} \begin{bmatrix}x(k+1)\\ y(k+1)\\ \theta(k+1)\end{bmatrix}=\begin{bmatrix}x(k)\\ y(k)\\ \theta(k)\end{bmatrix}+\Delta T\begin{bmatrix}v(k)\cos\theta(k)\\ v(k)\sin\theta(k)\\ \omega(k)\end{bmatrix}$$

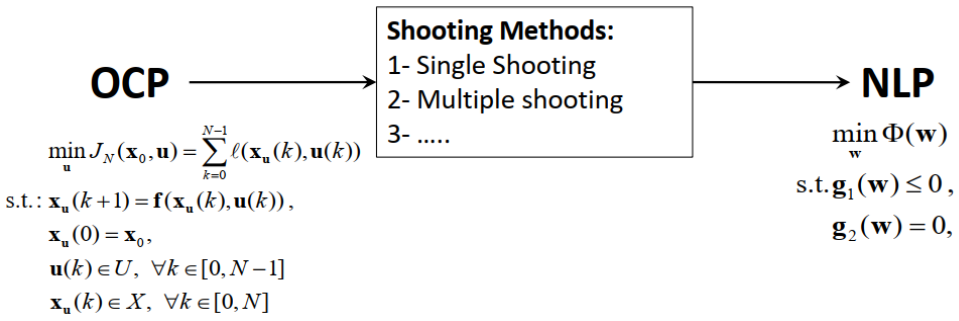
Then, the running cost of the system is defined where we are going to penalize the control action and the output states of the system.

$$\ell(\mathbf{x},\mathbf{u})=\left\|\mathbf{x}_{\mathbf{u}}-\mathbf{x}^{ref}\right\|_{\mathbf{Q}}^2+\left\|\mathbf{u}-\mathbf{u}^{ref}\right\|_{\mathbf{R}}^2$$

Then, the objective function is defined in order to formulate the optimal control problem:

$$\begin{aligned}&\underset{\mathbf{u}\,\text{admissible}}{\text{minimize}}\, J_N(\mathbf{x}_0,\mathbf{u})=\sum_{k=0}^{N-1}\ell(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k))\\ &\text{subject to}: \mathbf{x}_{\mathbf{u}}(k+1)=\mathbf{f}(\mathbf{x}_{\mathbf{u}}(k),\mathbf{u}(k)),\\ &\qquad\qquad \mathbf{x}_{\mathbf{u}}(0)=\mathbf{x}_0,\\ &\qquad\qquad \mathbf{u}(k)\in U,\ \ \forall k\in[0,N-1]\\ &\qquad\qquad \mathbf{x}_{\mathbf{u}}(k)\in X,\ \ \forall k\in[0,N]\end{aligned}$$

And finally, the non-linear programming approach was selected to execute the online optimization problem. The Single Shooting method was used to transform the optimal control problem to the non-linear programming problem.



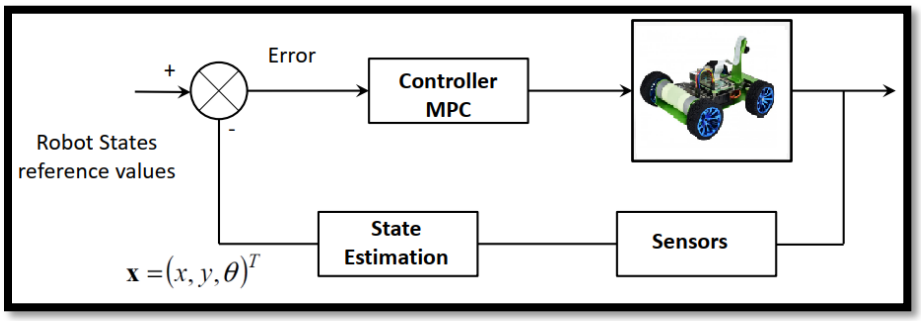


Figure 2.The block diagram of the system

2.3 MATLAB Script – Step by Step

The CasADi library was used to solve our Optimal Control Problem using non-linear programming approach.

Also, the MATLAB support package for raspberry pi was used to connect the script to R-Pi4 which is used on Donkey Car as its main control hardware.

1- First of all, change the MATLAB directory to the CasADi folder and then start a new .m file.
2- In order to start the wifi communication with R-Pi4, add the following lines to the file:

```
clc; clear all; close all
import casadi.*

mypi = raspi();
srv = servo (mypi, 26);
writePosition(srv, 90);

configurePin(mypi, 12, 'PWM');
configurePin(mypi, 21, 'PWM');

writePWMFrequency(mypi, 12, 2000);
writePWMFrequency(mypi, 21, 2000);

writePWMVoltage(mypi, 12, 0);
writePWMVoltage(mypi, 21, 0);
```

3- The sampling time, T, equals to 0.2, the prediction horizon, N, equals to 10, the maximum allowed speed is 1.8 m/s, and the maximum angular speed of the robot equals to pi/2.5 rad/s.

Continue to enter the following lines in to the code:

```
T = 0.2; % sampling time [s]
N = 15; % prediction horizon
rob_diam = 0.65;

v_max = 1.8;
v_min = -v_max;

omega_max = pi/2.5;
omega_min = -omega_max;
```

4- As it mentioned before, the CasADi library uses symbolic variables to construct the optimal control problem and then it convert this symbolic problem into the non-linear programming problem. So, the X position, the Y position, and the Theta angle of the robot is defined as the symbolic variables using CasADi special syntax.

Continue to enter the following lines into the code:

```
omega_max = pi/2.5;
omega_min = -omega_max;

x = SX.sym('x');
y = SX.sym('y');
theta = SX.sym('theta');
v = SX.sym('v');
omega = SX.sym('omega');
```

5- This control system consists of 2 control variables, v and omega, and three states, x and y and theta, which makes the system multi-input multi-output.
Continue to enter the following lines into the code:

```
states = [x;y;theta];
n_states = length(states);

controls = [v;omega];
n_controls = length(controls);
rhs = [v*cos(theta);v*sin(theta);omega]; % system r.h.s
```

6- To create the optimal control problem, it is necessary to define a function, f, which takes the control actions and calculates the states of the system. Also, the overall control actions over the optimization process, U, optimal control parameters, P, and state variables, X, need to be defined.
Continue to enter the following lines into the code:

```
f = Function('f',{states,controls},{rhs});
U = SX.sym('U',n_controls,N);
P = SX.sym('P',n_states + n_states);
X = SX.sym('X',n_states,(N+1));
```

7- the 'con' and 'st' cariables represent the control actions and state variables over the all prediction horizon. Using a loop, these variables were created.
Continue to enter the following lines into the code:

```
X(:,1) = P(1:3); % initial state
for k = 1:N
    st = X(:,k);
    con = U(:,k);
    f_value  = f(st,con);
    st_next  = st+ (T*f_value);
    X(:,k+1) = st_next;
end
```

8- Knowing the control actions and the optimal control parameters, the function, ff, will calculate the optimal trajectory. Continue to enter the following lines into the code:

```
ff=Function('ff',{U,P},{X});
```

9- The objective function was created over the prediction horizon. Continue to enter the following lines into the code:

```
obj = 0; % Objective function

Q = zeros(3,3);
Q(1,1) = 1;
Q(2,2) = 5;
Q(3,3) = 0.1; % weighing matrices (states)

R = zeros(2,2);
R(1,1) = 0.5;
R(2,2) = 0.05; % weighing matrices (controls)

for k=1:N
    st = X(:,k);
    con = U(:,k);
    obj = obj+(st-P(4:6))'*Q*(st-P(4:6)) + con'*R*con; % calculate obj
end
```

10- The constraints on states x and y: Continue to enter the following lines into the code:

```
g = [];  % constraints vector
for k = 1:N+1   % box constraints due to the map margins
    g = [g ; X(1,k)];   %state x
    g = [g ; X(2,k)];   %state y
end
```

11- At the next step, a struct is defined to create the parameters of non-linear programming problem. Also, CasADi uses the interior point optimization technique. So, it is necessaru to define its parameters as CasADi suggested. Continue to enter the following lines into the code:

```
OPT_variables = reshape(U,2*N,1);
nlp_prob = struct('f', obj, 'x', OPT_variables, 'g', g, 'p', P);

opts = struct;
opts.ipopt.max_iter = 100;
opts.ipopt.print_level =0;%0,3
opts.print_time = 0;
opts.ipopt.acceptable_tol =1e-8;
opts.ipopt.acceptable_obj_change_tol = 1e-6;

solver = nlpsol('solver', 'ipopt', nlp_prob,opts);
```

12- The input constraints and inequality constraints should be entered to the code as follows:

```
args = struct;
% inequality constraints (state constraints)
args.lbg = -2;  % lower bound of the states x and y
args.ubg = 2;   % upper bound of the states x and y

% input constraints
args.lbx(1:2:2*N-1,1) = v_min;
args.lbx(2:2:2*N,1)   = omega_min;
args.ubx(1:2:2*N-1,1) = v_max;
args.ubx(2:2:2*N,1)   = omega_max;
```

13- The MPC process starts at this point. So far, the optimal control problem and the non-linear programming problem were created. Know, it's the time to use the initial condition and created NLP to design a MPC controller for the robot. Continue to enter the following variables into the code:

```
t0 = 0;
x0 = [0 ; 0 ; 0.0];    % initial condition.[x y theta]
xs = [2; 2 ; 0]; % Reference Point
xx(:,1) = x0; % xx contains the history of states
t(1) = t0;
u0 = zeros(N,2);  % two control inputs
sim_tim = 20; % Maximum simulation time
mpciter = 0;
xx1 = [];
u_cl=[];
```

14- This is the main simulation loop which works as ling as the error is greater than 10^2 and the number of MPC steps is less than its maximum value.

```
main_loop = tic;
while(norm((x0-xs),2) > 1e-2 && mpciter < sim_tim / T)
    args.p   = [x0;xs]; % set the values of the parameters vector
    args.x0 = reshape(u0',2*N,1); % initial value of the optimization variables
    %tic
    sol = solver('x0', args.x0, 'lbx', args.lbx, 'ubx', args.ubx,...
            'lbg', args.lbg, 'ubg', args.ubg,'p',args.p);
    u = reshape(full(sol.x)',2,N)';
    ff_value = ff(u',args.p); % compute OPTIMAL solution TRAJECTORY
    xx1(:,1:3,mpciter+1)= full(ff_value)';

    u_cl= [u_cl ; u(1,:)];
    t(mpciter+1) = t0;
    % Deploy the Signal-------------------------
    writePosition(srv, 90-(u(1,2)*180/pi));
    writePWMVoltage(mypi, 12, abs(1.66*u(1,1)));
    writePWMVoltage(mypi, 21, abs(1.66*u(1,1)));
    %-----------------------------------------
    [t0, x0, u0] = shift(T, t0, x0, u,f); % get the initialization of the next
optimization step
    xx(:,mpciter+2) = x0;
    mpciter;
    mpciter = mpciter + 1;
    pause(0.2);
end
```

15-The results of the simulation:

```
% Results
figure(1);
plot(xx(1,:), xx(2,:), 'LineWidth', 2, 'color', 'red');
grid on
box on
title('The Simulated Trajectory for the Robot');
xlabel('X(m)');
ylabel('Y(m)');

figure(2);
subplot(211);
plot(u_cl(:,1), 'LineWidth', 2, 'color', 'blue');
grid on
box on
title('The Input Speed of the Robot');
xlabel('T(s)');
ylabel('V(m/s)');

% figure(3);
subplot(212);
plot(xx(3,:), 'LineWidth', 2, 'color', 'black');
grid on
box on
title('The Input Angle of the Robot');
xlabel('T(s)');
ylabel('Theta(m/s)');
```

2.4 The Simulation Results

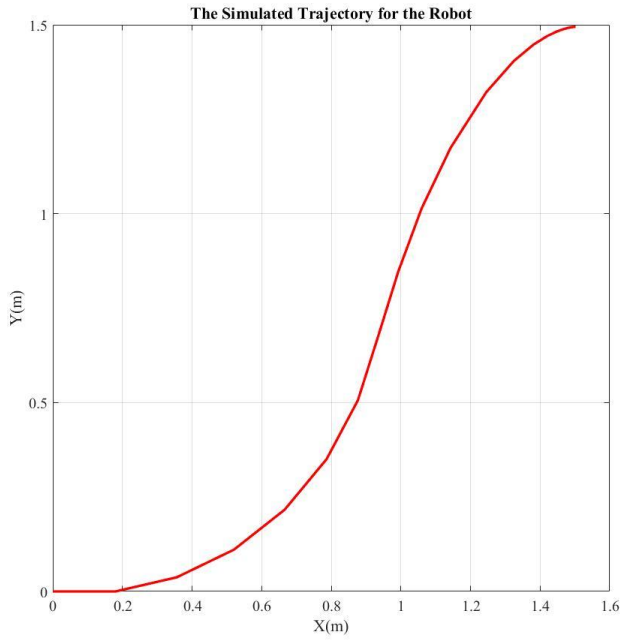


Figure 3. The simulation results coming from MPC controller using CasADi library

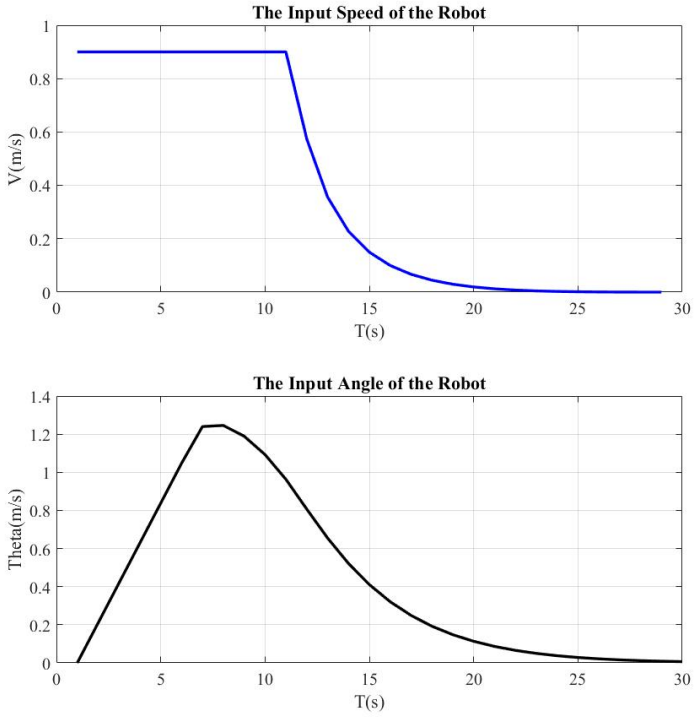


Figure 4. The calculated control actions while robot is moving from point A to point B

3 Model Predictive Control Implementation – No object avoidance algorithm is considered in this step

In this section, an obstacle avoidance algorithm in considered within the MPC controller. The robot starts from pint A and it's going to be driven to point B using the MPC controller while avoiding an object in the area. It is assumed that the robot is approximated by a circle as well as the obstacle is approximated by a circle. Technically, we don't want these two circles intersect.

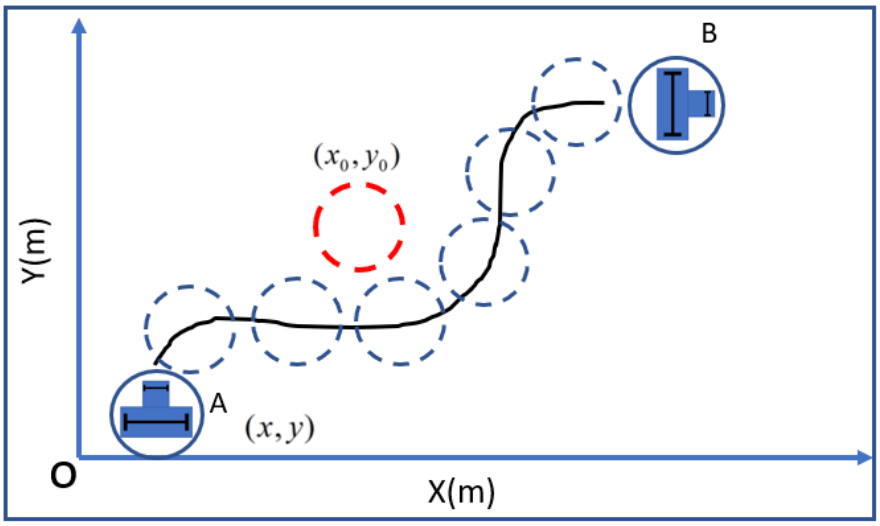


Figure 5. The Object Avoidance schematic which is going to be implemented as a constraint within the MPC control algorithm

So, basically, we have the radius r_r for the robot and the radius r_o for the obstacle. We would like to maintain lower bound for the Euclidean distance between the prediction and the obstacle position. Therefore, we need to impose the following path constrains.

$$\sqrt{(x-x_0)^2+(y-y_0)^2} \geq r_r + r_0$$
$$\sqrt{(x-x_0)^2+(y-y_0)^2} - (r_r + r_0) \geq 0$$
$$-\sqrt{(x-x_0)^2+(y-y_0)^2} + (r_r + r_0) \leq 0$$

It is assumed that this obstacle is static. So, x_o and y_o are static as well. The x and y will be the predicted states for the robot. As long as the x and y are changing, we always want to maintain the previous inequality statement.

So, technically, we will implement all things like it's done before in the previous section. The only difference is that we have another constrain within the MPC controller which is defined above. Please continue to enter the following lines into the code after step #10 and before step #11.

```
% Add constraints for collision avoidance
obs_x = 0.5; % meters
obs_y = 0.5; % meters
obs_diam = 0.3; % meters
for k = 1:N+1   % box constraints due to the map margins
    g = [g ; -sqrt((X(1,k)-obs_x)^2+(X(2,k)-obs_y)^2) + (rob_diam/2 + obs_diam/2)];
end
```

The next step is to add the upper and the lower bound, these constraints within the optimal control structure. According to the inequality constrains for the object avoidance algorithm, the upper bound is '0' and the lower bound the '-inf'. To do so, please substitute the step #12 by the following cod

```
% Results
figure(1);
plot(xx(1,:), xx(2,:), 'LineWidth', 2, 'color', 'red');
grid on
box on
hold on
xc = obs_x - obs_diam/2 :0.001: obs_x + obs_diam/2;
yc = sqrt((obs_diam/2).^2-(xc-obs_x).^2) + obs_y;
plot(xc, yc);
plot(xc, -yc+2*obs_y);
hold on
xpl = xx(1,:)';
ypl = xx(2,:)';
for i = 1:3:26
    xc1 = xpl(i) - rob_diam/2 :0.001: xpl(i) + rob_diam/2;
    yc1 = sqrt((rob_diam/2).^2-(xc1-xpl(i)).^2) + ypl(i);
    plot(xc1, yc1, 'color', 'black');
    hold on
    plot(xc1, - yc1+2*ypl(i), 'color', 'black');
    hold on
end

title('The Simulated Trajectory for the Robot While Avoiding a Static Object');
xlabel('X(m)');
ylabel('Y(m)');

figure(2);
subplot(211);
plot(u_cl(:,1), 'LineWidth', 2, 'color', 'blue');
grid on
box on
title('The Input Speed of the Robot');
xlabel('T(s)');
ylabel('V(m/s)');

% figure(3);
subplot(212);
plot(xx(3,:), 'LineWidth', 2, 'color', 'black');
grid on
box on
title('The Input Angle of the Robot');
xlabel('T(s)');
ylabel('Theta(m/s)');
```

3.1 The Simulation Results

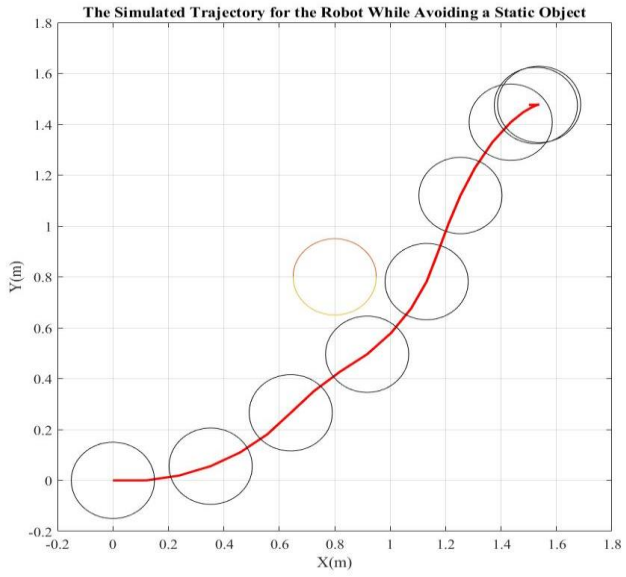


Figure 6. The simulation results coming from MPC controller using CasADi library while avoiding a static object
Object Location: (x = 0.5, y = 0.5)

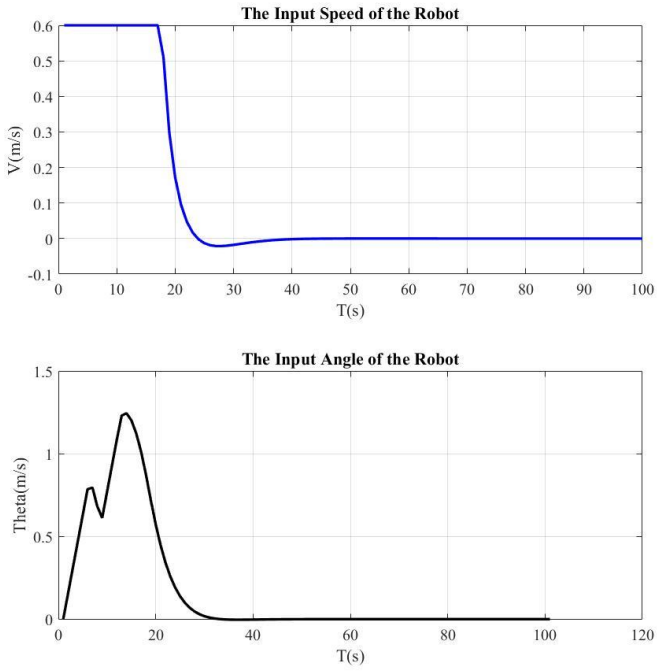


Figure 7. The calculated control actions while robot is moving from point A to point B while avoiding a static object
Object Location: (x = 0.5, y = 0.5)

4 References

All the references are attached to this report.