Kinematic model of Otbot

This live script develops the kinematic model of Otbot.

**Table of Contents**

[Introduction to the robot](#gjdgxs)  [Reference frames and vector bases](#30j0zll)  [Configuration and state coordinates](#1fob9te) [Kinematic constraints imposed by the rolling contacts](#3znysh7)  [Kinematic constraints of the differential drive](#2et92p0)  [Kinematic constraints of the whole robot](#tyjcwt) [Solution to the instantaneous kinematic problems](#3dy6vkm)  [Forward problem](#1t3h5sf)  [Inverse problem](#4d34og8) [The kinematic model in control form](#2s8eyo1) [Save matrices](#17dp8vu) [Print ellapsed time](#3rdcrjn)

We start with a few initializations:

% Clear all variables, close all figures, and clear the command window

clearvars

close all

clc

% Start stopwatch

tic

% Display the matrices with rectangular brackets

sympref('MatrixWithSquareBrackets',true);

% Turn off abbreviated output format (avoid Matlab's own substitution of long expressions)

sympref('AbbreviateOutput',false);

% Symbolic variables to be used (see the figures and explanations below)

syms l\_1 l\_2 r % Pivot offset, semiaxis length, and wheel radius (constants)

syms theta theta\_dot % Absolute angular velocity of the chassis

syms v % Velocity of the chassis point M along direction 1'

syms varphi\_dot\_l % Angular velocity of left wheel

syms varphi\_dot\_r % Angular velocity of right wheel

syms a b % Absolute coordinates of M

syms a\_dot b\_dot % Absolute velocity components of M

syms x y % Absolute coordinates of the pivot point P

syms x\_dot y\_dot % Absolute velocity components of P

syms alpha alpha\_dot % Absolute angle and angular velocity of the platform

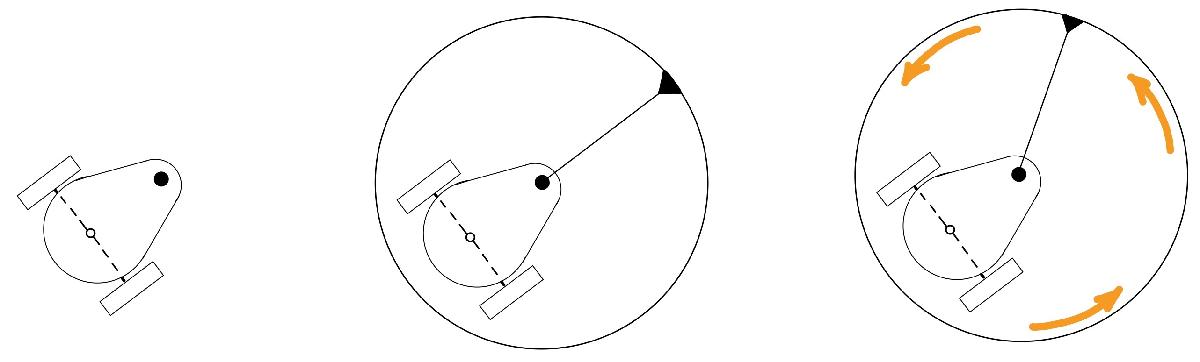
syms varphi\_p % Pivot angle

syms varphi\_dot\_p % Angular velocity of the pivot joint

## Introduction to the robot

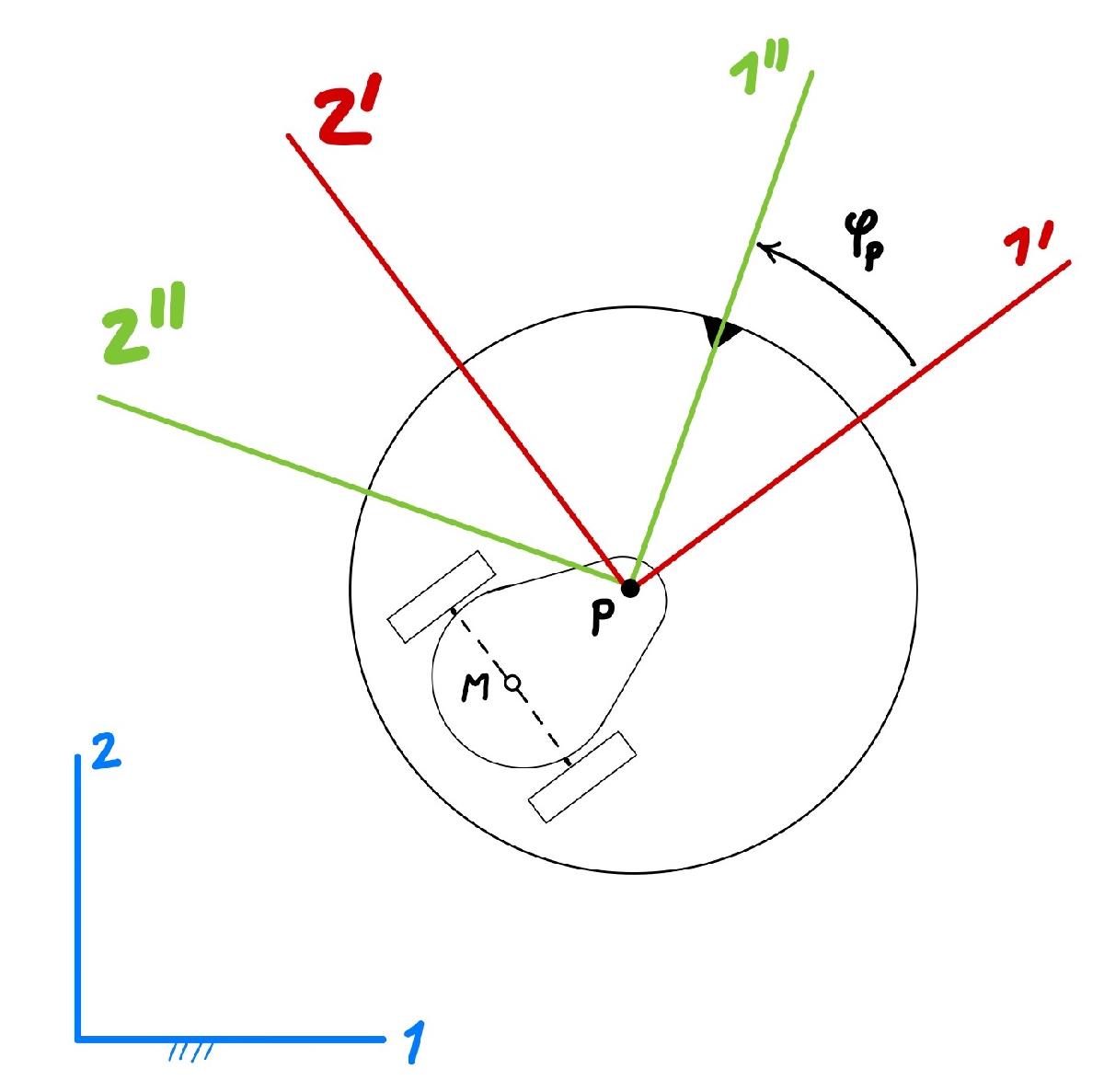
### Reference frames and vector bases

The following pictures describe the kinematic structure of Otbot:



The robot chassis (left picture) is a classical differential drive robot whose two wheels are powered by DC motors. Passive caster wheels below the chassis (not drawn) impede the tumbling of the robot. A circular platform is mounted on top of the chassis (center picture) which can rotate relative to it (right picture). The rotation is performed by means of a pivot joint (black dot) that is controlled by another DC motor. Note that the pivot joint has an offset relative to the wheels' axis. This offset will allow the circular platform to behave like an omnidirectional vehicle in the plane.

To obtain the kinematic and dynamic models of the robot, we will use the following reference frames:

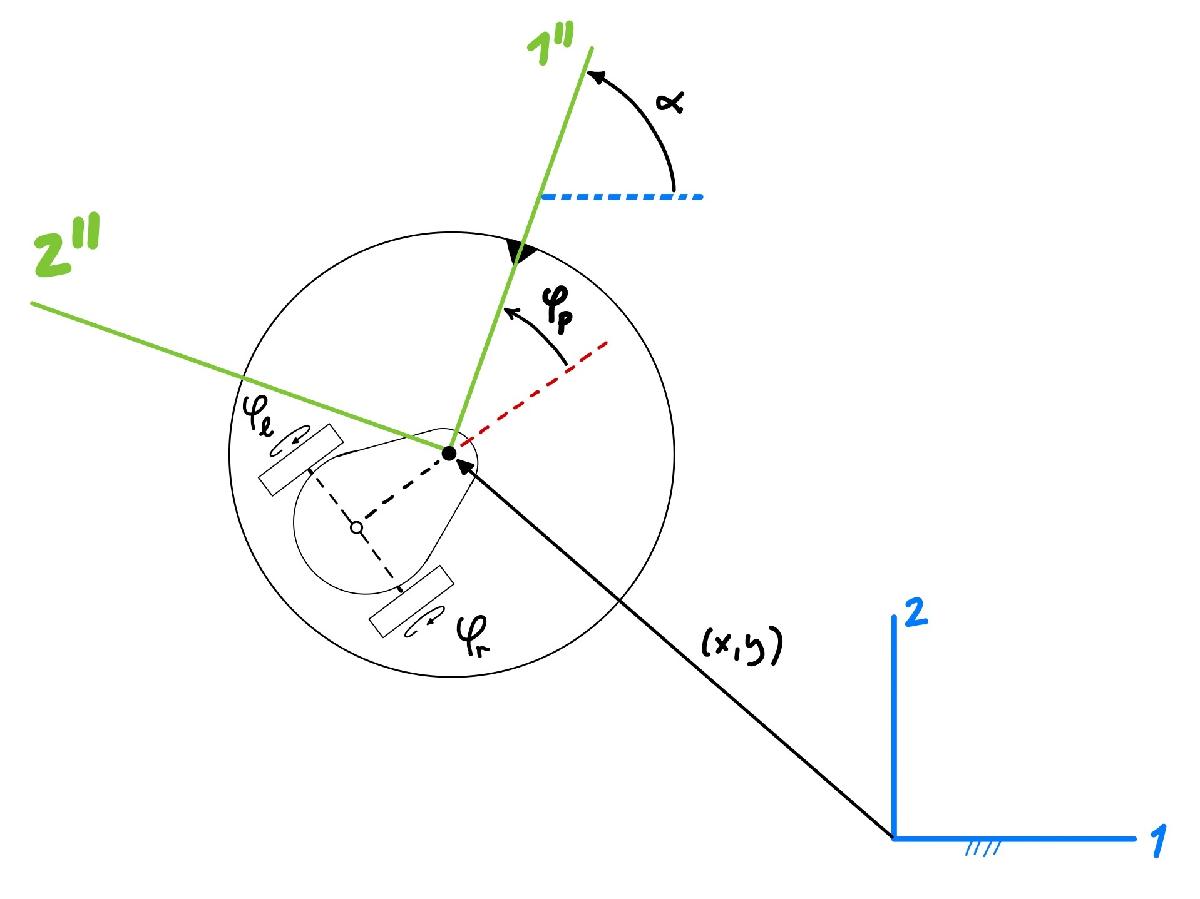


The blue frame is the absolute frame fixed to the ground. The red and green frames are fixed to the chassis and the platform, respectively. Point M is the midpoint of the wheels axis and point P is the location of the pivot joint. The red axis 1' is always aligned with M and P. The angle between the green and red frames coincides with the pivot joint angle .

Each frame has a vector basis attached to it, whose unit vectors are directed along the frame axes. The three bases will be referred to as B = {1,2,3}, B' = {1', 2', 3'} and B" = {1", 2", 3"} respectively.

### Configuration and state coordinates

The robot configuration can be described by the following six coordinates: the absolute position  of the pivot joint, the absolute angle *α* of the platform, the pivot angle , and the angles of the right and left wheels,  and , relative to the chassis.



The robot configuration is thus given by



and the time derivative of  provides the robot velocity



Therefore, the robot state will be given by



## Kinematic constraints imposed by the rolling contacts

We will see that the coordinates of  are not independent. The rolling contacts of the wheels impose a kinematic constraint of the form



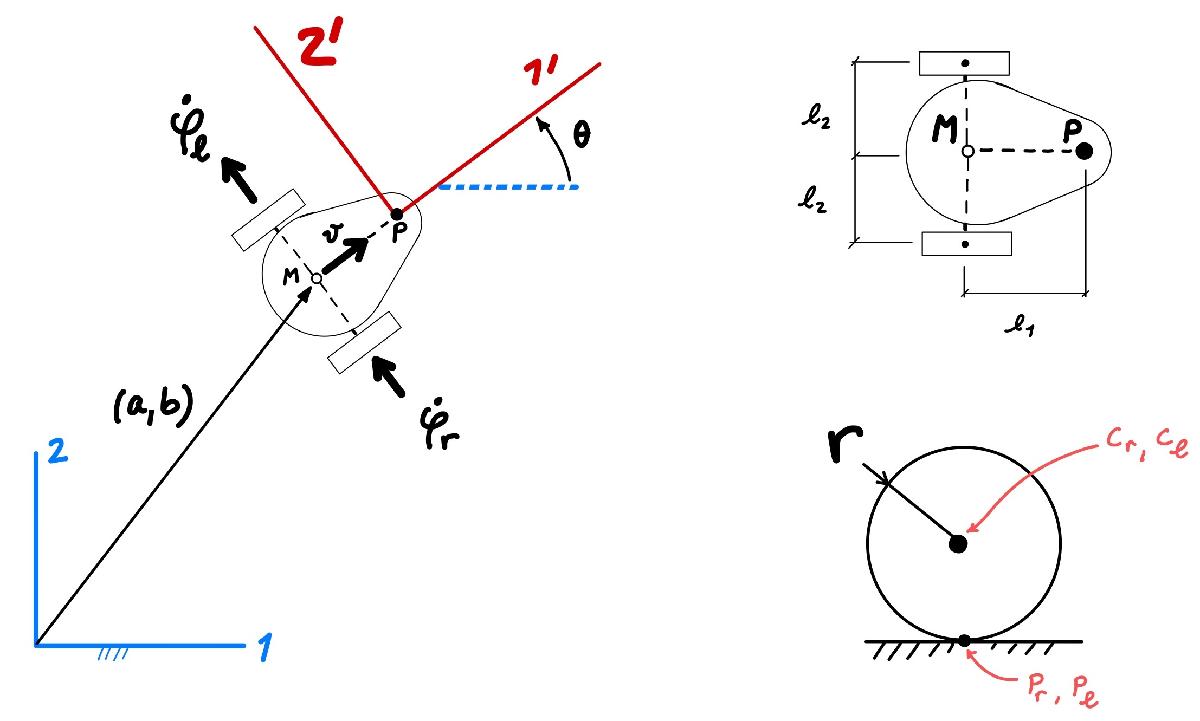
where  is a  Jacobian matrix. We derive such a constraint in two steps:

1. We obtain the kinematic constraints imposed by the wheels.
2. We rewrite these constraints using the  variables only.

We next see these steps in detail. On our derivations, we use  to denote the velocity of a point *Q* of the robot. The basis in which  is expressed will be mentioned explicitly, or understood by context.

### Kinematic constraints of the differential drive

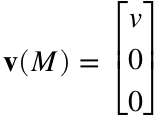
For the moment, let us neglect the platform and focus our attention on the chassis:



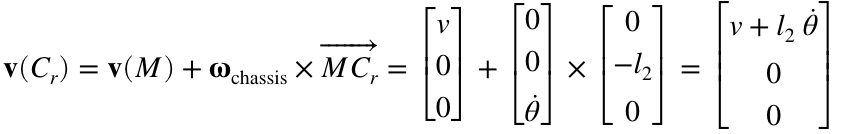
The chassis pose is given by the position vectorof point M, and by the orientation angle *θ*. M is the midpoint of the wheels axis, and P is the location of the pivot joint. We use  and  to refer to the pivot offset from M and the half-length of the wheels axis, respectively. Also,  and  denote the centers of the right and left wheels, and  and  are the contact points of such wheels with the ground. The two wheels have the same radius *r*.

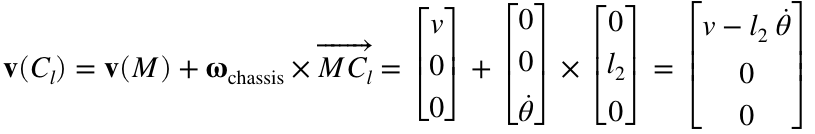
The rolling contact constraints of the chassis can be found by computing the velocities of  and  in terms of , , , , and  (i.e., as if the robot were a floating kinematic tree) and forcing these velocities to be zero (as the wheels do not slide when placed on the ground).

To obtain  and , note first that the velocity of *M* can only be directed along axis 1', since lateral slipping is forbidden under perfect rolling. Therefore in B' = {1',2',3'}



Also note that,



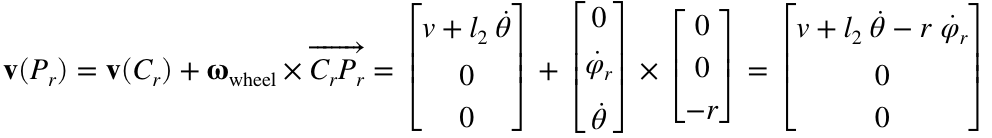


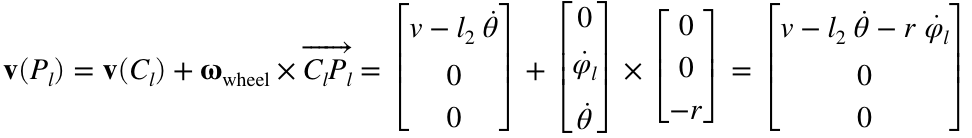
or, in Matlab syntax:

v\_of\_Cr = [v;0;0] + cross( [0;0;theta\_dot], [0;-l\_2;0] ); % Velocity of Cr

v\_of\_Cl = [v;0;0] + cross( [0;0;theta\_dot], [0; l\_2;0] ); % Velocity of Cl

The velocities of the ground contact points are thus given by





i.e.,

v\_of\_Pr = v\_of\_Cr + cross( [0; varphi\_dot\_r; theta\_dot], [0; 0; -r]); % Velocity of Pr

v\_of\_Pl = v\_of\_Cl + cross( [0; varphi\_dot\_l; theta\_dot], [0; 0; -r]); % Velocity of Pl

Since  and  do not slip,  and  must be zero, which gives the two fundamental constraints of the robot:

eqn1 = (v\_of\_Pr(1) == 0);

eqn2 = (v\_of\_Pl(1) == 0);

disp(eqn1); disp(eqn2);





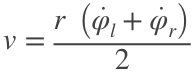
It is now easy to solve for *v* and  in these two equations:

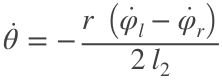
[solved\_v,solved\_thetadot] = solve([eqn1,eqn2],[v,theta\_dot]);

eqn3 = (v==simplify(solved\_v));

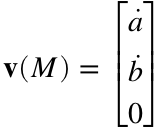
eqn4 = (theta\_dot==simplify(solved\_thetadot));

disp(eqn3); disp(eqn4);

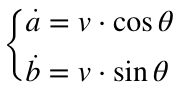


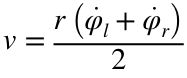
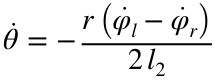


Note from the previous figure that in the basis B = {1,2,3}



and that it must be



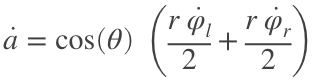
By substituting  in these two equations, and also considering , we obtain the system:

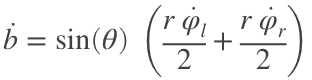
eqn5 = a\_dot == solved\_v \* cos(theta);

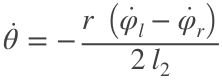
eqn6 = b\_dot == solved\_v \* sin(theta);

eqn7 = eqn4;

disp(eqn5); disp(eqn6); disp(eqn7);





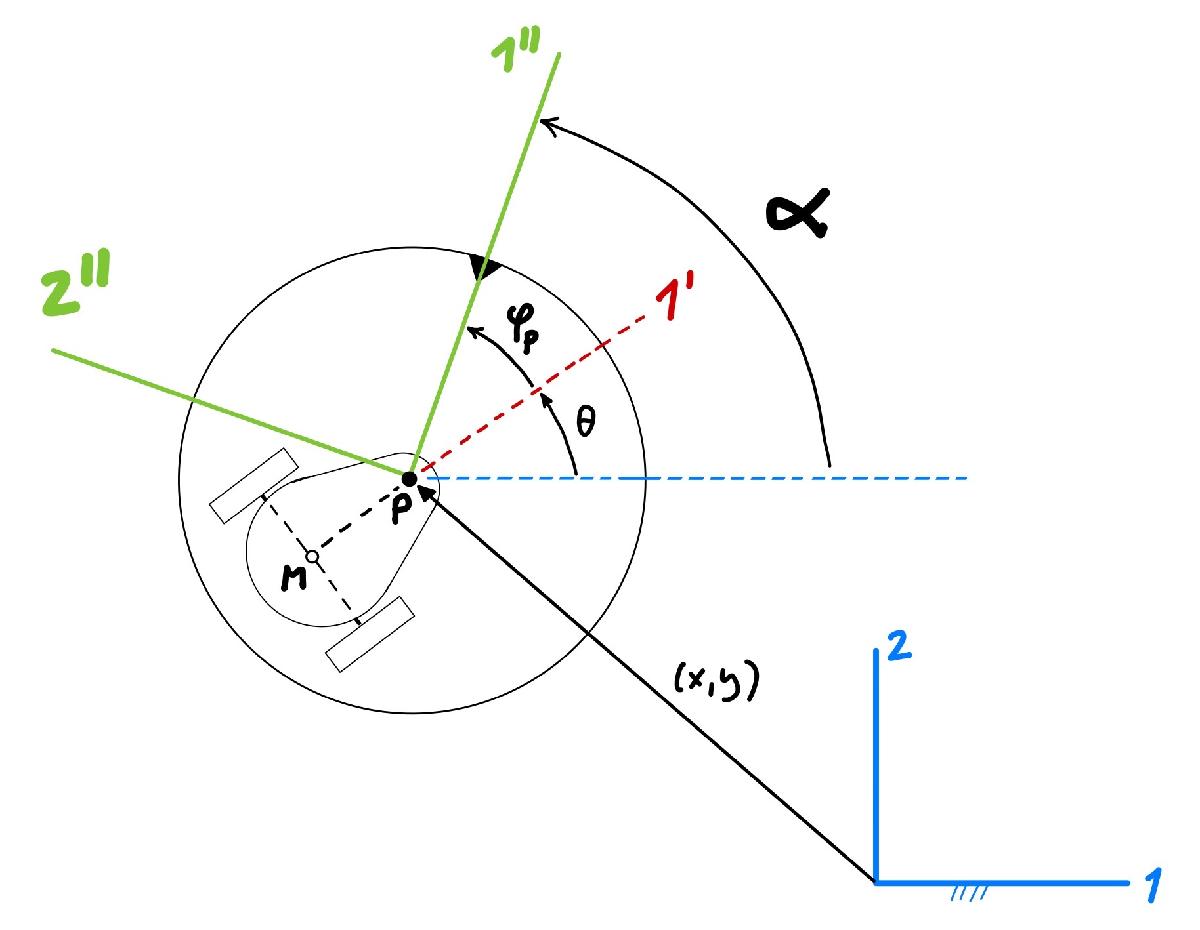


This system provides the forward kinematic solution for the differential drive. It can also be viewed as the system of kinematic constraints that express the rolling contact constraints.

### Kinematic constraints of the whole robot

To obtain the kinematic constraints of Otbot itself, we just need to rewrite the previous system using the variables in . This entails substituting *θ*, , , and  by their expressions in terms of the coordinates in .

By looking at the figure



it is clear that

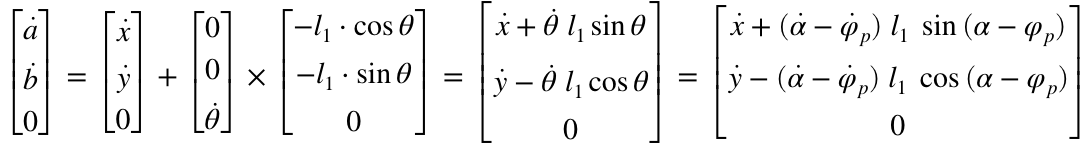




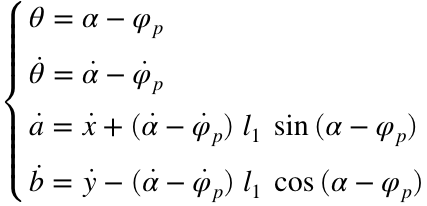
and we also see that

,

so using B = {1, 2, 3} we can write



In sum we have the relationships



which give the desired values of *θ*, , , and  in terms of  and . We thus can substitute these expressions in eqn5, eqn6, and eqn7 above to obtain eqn8, eqn9, and eqn10:

theta\_value = alpha - varphi\_p;

theta\_dot\_value = alpha\_dot - varphi\_dot\_p;

a\_dot\_value = x\_dot + (alpha\_dot - varphi\_dot\_p) \* l\_1 \* sin(alpha-varphi\_p);

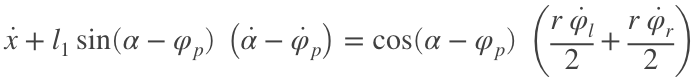
b\_dot\_value = y\_dot - (alpha\_dot - varphi\_dot\_p) \* l\_1 \* cos(alpha-varphi\_p);

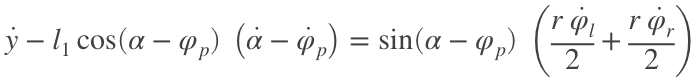
eqn8 = subs(eqn5,[a\_dot,theta],[a\_dot\_value,theta\_value]);

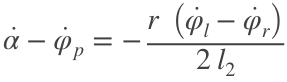
eqn9 = subs(eqn6,[b\_dot,theta],[b\_dot\_value,theta\_value]);

eqn10 = subs(eqn7,theta\_dot,theta\_dot\_value);

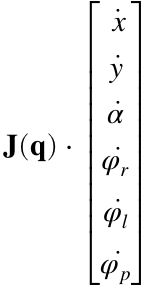
disp(eqn8); disp(eqn9); disp(eqn10);







In matrix form, these equations can be written as

= **0**,

where  is the  matrix

J\_of\_q = equationsToMatrix(...

[eqn8,eqn9,eqn10],...

[x\_dot,y\_dot,alpha\_dot,varphi\_dot\_r,varphi\_dot\_l,varphi\_dot\_p])

J\_of\_q =

## Solution to the instantaneous kinematic problems

Let  and . These vectors provide the task and joint space variables of the robot. Their time derivatives  and  are called the motor speeds and the platform twist, respectively

This section solves the following two problems:

* The forward instantaneous kinematic problem (**FIIKP**): Obtain  as a function of .
* The inverse instantaneous kinematic problem (**IIKP**): Obtain  as a function of .

It is also shown that the IIKP is always solvable, irrespective of the robot configuration. This implies that Otbot's platform can move omnidirectionally in the plane.

### Forward problem

We only have to isolate , , and  from the earlier equations eqn8, eqn9, and eqn10 defining :

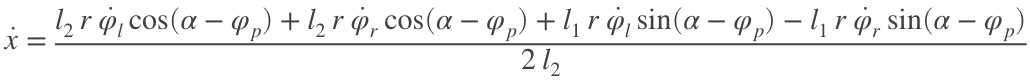
[solved\_xdot, solved\_ydot, solved\_alphadot] = solve([eqn8,eqn9,eqn10],[x\_dot,y\_dot,alpha\_dot]);

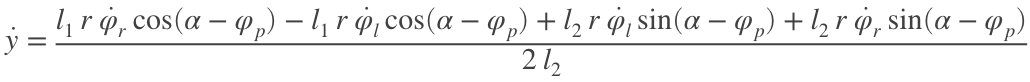
eqn11 = (x\_dot == solved\_xdot);

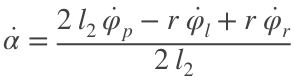
eqn12 = (y\_dot == solved\_ydot);

eqn13 = (alpha\_dot == solved\_alphadot);

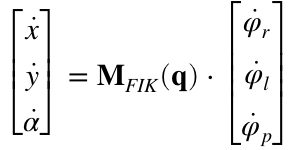
disp(eqn11); disp(eqn12); disp(eqn13);







These equations directly provide  as a function of . This function can be expressed as

,

where  is the following  matrix:

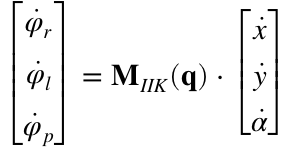
MFIK\_of\_q = equationsToMatrix([solved\_xdot, solved\_ydot, solved\_alphadot],...

[varphi\_dot\_r, varphi\_dot\_l, varphi\_dot\_p])

MFIK\_of\_q =

### Inverse problem

We now wish to find  as a function of . Clearly, this function is given by

,

where .

Note that the determinant of  is

detMFIK = simplify(det(MFIK\_of\_q))

detMFIK =

Since *r*, , and  are all positive in Otbot, we see that  always exists, so the IIKP is solvable in all configurations of the robot. An important consequence is that the platform will be able to move under any twist  in the plane. In other words, it will be an omnidirectional platform.

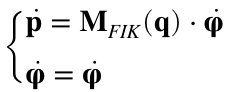
 has the expression:

MIIK\_of\_q = simplify(inv(MFIK\_of\_q))

MIIK\_of\_q =

## The kinematic model in control form

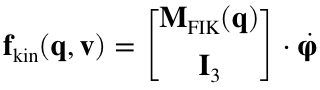
These equations



can be written in the usual form used in control engineering,

,

where  and

.

Notice that in this model  plays the role of the control actions, which are motor velocities in this case.

We could use this model for trajectory planning already. In doing so, we would neglect the system dynamics, but a strong point is the model simplicity, which leads to faster planners and controllers. Moreover, the model would be sufficient if the commanded velocities  where easy to control (e.g., if  is smooth enough for the motors at hand). The model would also be helpful to perform early tests of a planner under development, or to compute approximate trajectories to be later enhanced by an optimizer (one using the full dynamical model for example).

## Save matrices

save('forward\_inverse\_jacobians.mat',"MIIK\_of\_q","MFIK\_of\_q")

save('constraint\_jacobian.mat',"J\_of\_q")

## Print ellapsed time

toc

Elapsed time is 2.446570 seconds.